

INTEGRATED MODELING TOOL FOR PERFORMANCE ENGINEERING OF COMPLEX COMPUTER SYSTEMS

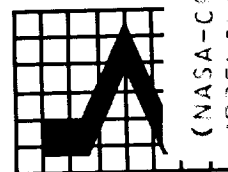


**NAS7-995 FINAL REPORT
DELIVERABLE 0002**

29 June 1989

NASA

**NATIONAL AERONAUTICS
AND SPACE ADMINISTRATION**



**ADVANCED SYSTEM
TECHNOLOGIES, INC.**

(NASA-CR-190881) INTEGRATED
MODELING TOOL FOR PERFORMANCE
ENGINEERING OF COMPLEX COMPUTER
SYSTEMS Final Report, Mar. 1987 -
29 Jun. 1989 (Advanced System
Technologies) 231 p

Copyright 1989, Advanced System Technologies, Inc.

This work was sponsored by the U.S. Government under NASA Contract NAS7-995. This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7025 (APR 1984).

SBIR RIGHTS NOTICE

This data is furnished with SBIR rights under NASA Contract No. NAS7-995. For a period of 2 years after acceptance of all items to be delivered under this contract, the Government agrees to use this data for Government purposes only, and it shall not be disclosed outside the Government during such period without permission of the Contractor, except that, subject to the foregoing use and disclosure prohibitions, such data may be disclosed for use by support contractors. After the aforesaid 2-year period, the Government has a royalty-free license to use, and to authorize others use on its behalf, this data for Government purposes; but is relieved of all disclosure prohibitions and assumes no liability for unauthorized use of this data by third parties. This Notice shall be affixed to any reproductions of this data, in whole or part.

**INTEGRATED MODELING TOOL FOR PERFORMANCE
ENGINEERING OF COMPLEX COMPUTER SYSTEMS**

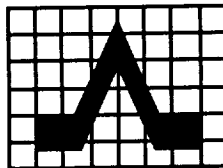


**NAS7-995 FINAL REPORT
DELIVERABLE 0002**

29 June 1989

Prepared by:

**Gary Wright
Duane Ball
Susan Hoyt
Oscar Steele**



**ADVANCED SYSTEM
TECHNOLOGIES, INC.**

**5113 Leesburg Pike,
Suite 514
Falls Church, VA 22041
(703) 845-0040**

Acknowledgements

The authors gratefully acknowledge the technical direction and support of Dr. Dan Erickson and Mr. Blair Lewis of the Jet Propulsion Laboratory. In addition, the comments and suggestions for prototype enhancements made by other JPL staff significantly contributed to the functionality of the delivered prototype.

Table of Contents

1.0	Project Summary	1
2.0	Background	2
3.0	Project Objectives	4
4.0	Accomplishments	5
4.1	Visual Interface	6
4.2	Translator	10
4.3	Simulator	11
4.4	Statistics Package	11
4.5	Software Interoperability	11
5.0	Results -- Sample PEDESTAL Models	14
5.1	Shared Resource Task Response Time Model	14
5.2	Telemetry Processing System	16
6.0	Conclusions and Recommendations	21
	Appendix -- PEDESTAL Preliminary User's Guide	A-1

1.0 Project Summary

Timely responses to stimuli are critical requirements for complex, real-time computer systems. To ensure these critical requirements are satisfied, tools and techniques capable of predicting compliance with response time specifications are needed. Unfortunately, application of these specialized computer performance engineering tools is either not applied or is misapplied during system design and development. As a result, systems often fail to meet their performance requirements and costly, timely redesign efforts are required to bring the system into compliance. To overcome this lack or misapplication of performance engineering, tools are needed which (1) require less specialized expertise to use and (2) can be applied throughout the design and development phases. That is, performance engineering tools must be used by the designers rather than by specialized performance engineering groups which are typically an adjunct to the design effort and therefore their models are reactive rather than proactive design aids.

There were two technical objectives for the Phase II research and development effort.

1. Develop an evaluation version of a graphical, integrated modeling language according to the specification resulting from the Phase I research.
2. Determine the degree to which the language meets its objectives by evaluating ease of use, utility of two sets of performance predictions, and the power of the language constructs.

The technical approach followed to meet these objectives was to design, develop, and test an evaluation prototype of a graphical, performance prediction tool. The utility of the prototype was then evaluated by applying it to a variety of test cases found in the literature and in AST case histories. Numerous models were constructed and successfully tested; the results of two such test cases are described in Section 5 of this report.

The major conclusion of this Phase II SBIR research and development effort is that complex, real-time computer systems can be specified in a non-procedural manner using combinations of icons, windows, menus, and dialogs. Such a specification technique provides an interface that system designers and architects find natural and easy to use. In addition, PEDESTAL's multi-view approach provides system engineers with the capability to perform the trade-offs necessary to produce a design that meets timing performance requirements. Sample system designs analyzed during the development effort showed that models could be constructed in a fraction of the time required by non-visual system design capture tools.

The potential applications of a graphical, integrated modeling tool are numerous. Architects and designers responsible for systems whose compliance with response times is critical and whose complexity requires analyses by modeling tools will greatly benefit from the proposed language. The tool supports incremental specification of design detail from high-level to low-level. Not only will the tool predict performance but it the simulator can also identify potential design defects (e.g., deadlocks and race conditions). Because (1) the system description language embodies a wide variety of constructs to represent software flows and (2) these constructs are easily entered by users, the graphical, integrated modeling language will be widely used and as a result, developed systems will more likely meet their specified performance criteria. As a result, costly redesign efforts will be avoided and, more importantly, systems that will fail to meet their mission requirements will not be deployed.

2.0 Background

Timely responses to stimuli are critical requirements for complex, real-time computer systems. To ensure these critical requirements are satisfied, tools and techniques capable of predicting compliance with response time specifications are needed. Unfortunately, application of these specialized computer performance engineering tools is either not applied or is misapplied during system design and development. As a result, systems often fail to meet their performance requirements and costly, timely redesign efforts are required to bring the system into compliance.

To overcome this lack or misapplication of performance engineering, tools are needed which (1) require less specialized expertise to use and (2) can be applied throughout the design and development phases. That is, performance engineering tools must be used by the designers rather than by specialized performance engineering groups which are typically an adjunct to the design effort and therefore their models are reactive rather than proactive design aids.

In addition to not supporting designers, traditional tools do not provide views of the system that enable trade-offs to be easily evaluated. In particular, most tools do not provide views that enable functional partitioning, functional allocation, data allocation, and operating system trade-offs to be easily performed. As a result, these trade-offs that are critical in the design of distributed systems are seldom evaluated.

Early in the design effort, many options often are evaluated and a few candidate alternatives are identified for more detailed analysis. Developing a tool that supports high-level rapid and low-level accurate design evaluations would provide designers with a tool that could be used during all phases of the design.

The development of such a timing performance prediction tool was the subject of this Phase II research and development effort. The goals of the development effort were to provide designers with a prediction tool with the following unique features:

- Single description of a computer system that captures the data required by both analytic and simulation modeling techniques.
- Graphical specification of hardware connectivity including dynamic and fixed path routing.
- Hardware node constructs that facilitate the specification and display of complex hardware topologies.
- Language constructs to fully characterize the behavior of concurrent software processes including: control flow arrows that capture all of the complexities involved in passing control from one module to another and software specification which is independent of the module-to-task and task-to-processor allocations.
- Ease in specifying resource management policies and their execution and transmission overheads.

- Sophisticated Human-Computer Interface (HCI) concepts that provide icons, menus, and dialogs specifically tailored to describing computer systems; support the operational use by system architects and designers; and support use by both novice and experienced users.

The combination of these features in a single tool will significantly advance the capabilities of currently available performance prediction tools and will provide tool users with the following benefits:

- Putting performance engineering tools in designers' hands - no longer will performance modeling be divorced from the design team. Designers will be able to immediately assess the timing performance impact of a pending design decision.
- Providing multiple scenarios to be analyzed prior to commitment to a final design thus ensuring that the system will perform as desired prior to deployment.
- Provide highly accurate performance predictions so that the complexities of embedded, real-time systems can be adequately addressed.
- Detect logical as well as performance bottlenecks - the discrete-event simulator can be used to detect such design flaws as deadlocks and race conditions.
- Reduce the cost of performance engineering by providing a single tool with the above characteristics.

3.0 Project Objectives

The overall objective of the Phase II research and development effort is to develop an evaluation prototype of a performance engineering tool that can be easily and effectively used to ensure that systems will meet their specified performance requirements. Ensuring compliance with requirements will avoid costly and lengthy redesign efforts, and, more importantly, avoid deployment of systems that fail to perform during critical missions.

There were two technical objectives for the Phase II research and development effort.

1. Develop an evaluation version of a graphical, integrated modeling language according to the specification resulting from the Phase I research.
2. Determine the degree to which the language meets its objectives by evaluating ease of use, utility of two sets of performance predictions, and the power of the language constructs.

The technical approach followed to meet these objectives was to design, develop, and test an evaluation prototype of a graphical, performance prediction tool. The utility of the prototype was then evaluated by applying it to a variety of test cases found in the literature and in AST case histories. Numerous models were constructed and successfully tested; the results of two such test cases are described in Section 5.

4.0 Accomplishments

The major accomplishment of the Phase II effort was the development of an evaluation prototype tool, called PEDESTAL, which was designed to assist system designers and architects in performance engineering of complex, real-time systems. The etymology for PEDESTAL is shown in Figure 4.0-1.

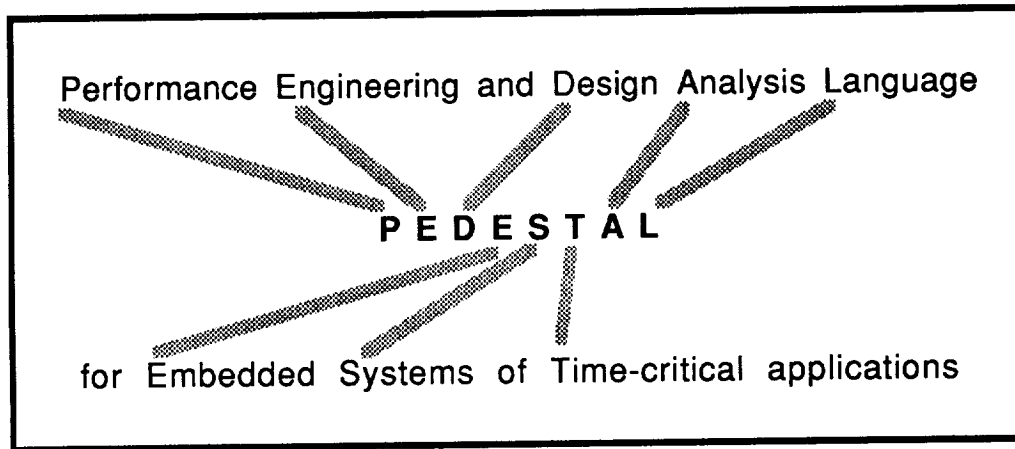


Figure 4.0-1: The PEDESTAL prototype is a performance engineering tool for time-critical applications

As shown in Figure 4.0-2, the PEDESTAL prototype consists of a visual programming interface, a translator, and a simulator. The PEDESTAL code module is approximately 428K bytes. As reference points, Word 3.01 is 350K; EXCEL is 385K; MacDraw II is 468K; and Pagemaker is 572K. The visual interface allows the user to describe the computer system using the mouse

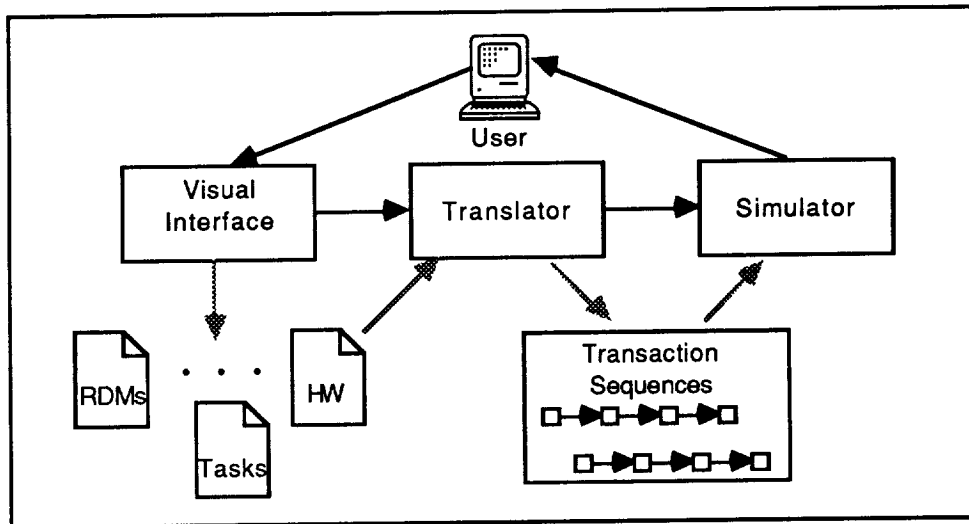


Figure 4.0-2: Three principal PEDESTAL components provide its functionality

and keyboard. This design information is stored in data structures managed by the interface software. Once the user has completed the system description, the RUN command invokes the

translator which converts the information in the interface data structures to sequences of operations to be performed on the transactions as they flow through the simulator.

4.1 Visual Interface - System Description Capability

Using the technology (windows, icons, menus, dialogs, etc.) offered by the Macintosh computer and system software, the user enters the information necessary to represent those system design features that can potentially impact timing performance (e.g., response time, throughput, utilization). The manner in which the user enters this information is described in detail in the PEDESTAL User's Guide which has been included as an Appendix to this report. A top-level summary of the design information that can be captured by PEDESTAL is depicted in Figure 4.1-1. A brief discussion of each of the design information categories follows.

System Requirements

A system is designed to satisfy a set of requirements. The timing requirements that influence design are the workloads that the system must process while meeting specified response time requirements.

Workload Parameters - Workload parameters are specified for each stimulus that arrives from a source external to the system being designed as well as for internally generated stimuli. Multiple workloads may be assigned to a single stimulus. For each workload, the key parameters that PEDESTAL captures are arrival rate (including rate distribution), number of workload items batched upon arrival, and workload priority. Workload arrivals may be initiated and terminated based on the time value of the simulated clock. As workloads flow through the simulated software, branching decisions and hardware scheduling decisions may be based upon workload type.

Performance Requirements - PEDESTAL provides a way to instrument the model to collect response times that correspond to those required by the system specifications. Start and stop delimiters can be specified at module entry and exit points. Finer response time granularity can be specified at resource demand initiation and termination points.

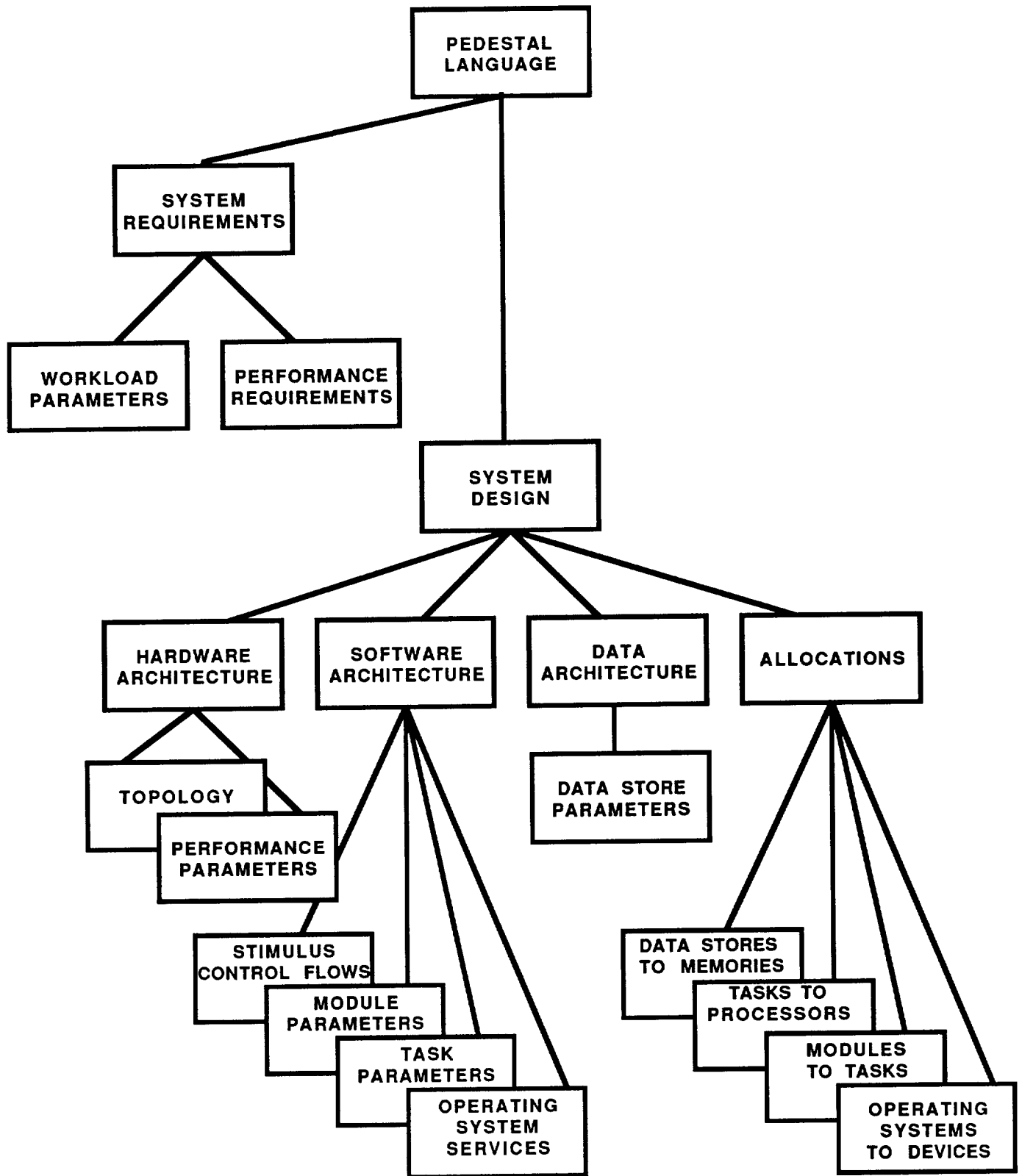


Figure 4.1-1: PEDESTAL language components capture all design features that impact performance

System Design

The first level of the system design is decomposed into hardware, software, and data architectures and the allocations that bind the software and data to the hardware.

Hardware Architecture - Specification of the hardware is supported by PEDESTAL's Topology and Performance Parameters language components. As shown in Figure 4.1-2, the hardware topology (number, type [processor, memory, bus], and physical connectivity of the hardware devices) is specified graphically by placing icons on a hardware window and connecting them via a series of line segments. An icon with a drop shadow depicts multiple servers (e.g., WS5-9 represents 5 workstations).

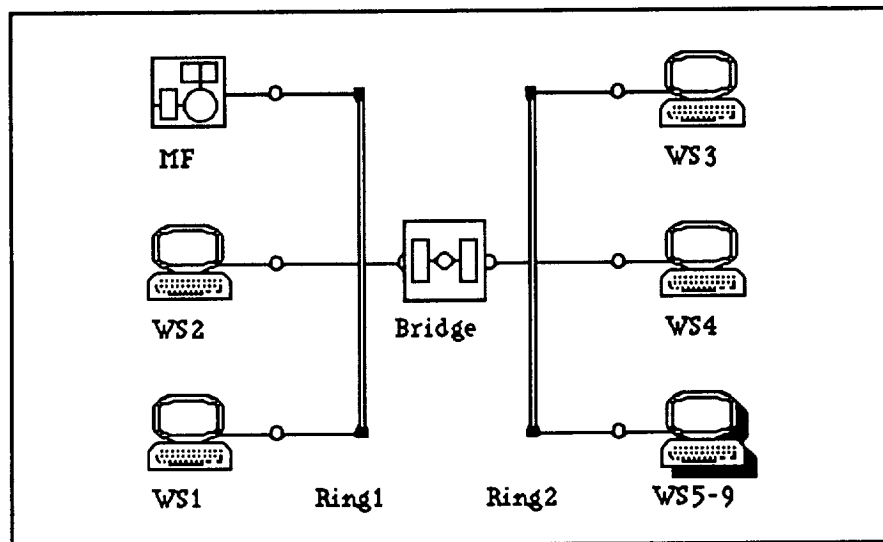


Figure 4.1-2: PEDESTAL's hardware window graphically depicts topology

The hardware performance parameters allow for specification of characteristics (e.g., instruction execution and data transmission rates, memory sizes) that determine device rates and capacities.

Software Architecture - Specification of the software architecture is supported by PEDESTAL's Stimulus Control Flows, Module Parameters, Task Parameters, and Operating System Services.

Application software is represented in PEDESTAL by two constructs: tasks and modules. Tasks represent the units of software concurrency and are the "dispatchable" software units. Modules are "atomic" units of software that are activated as a unit by the control logic within a task in a mutually exclusive manner (no two modules within a task may process concurrently).

The application software is represented as directed sequences of module executions called Stimulus Control Flows (SCFs). An SCF specifies the time-ordered flow of control of a stimulus from module to module. An SCF is initiated by the arrival of a stimulus and as shown in Figure 4.1-3, is constructed by placing SCF initiator, module, and terminator icons on a software window and connecting the icons with arrows. Drawing an arrow from one module to another represents passing control of processing the stimulus from one module to another. In most cases, data will also be passed along with control of the stimulus.

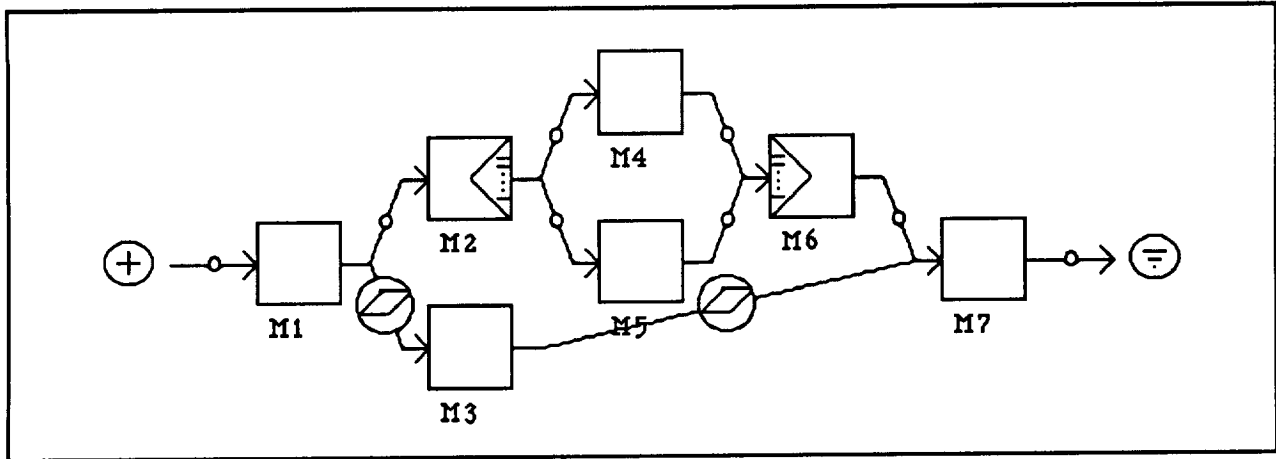


Figure 4.1-3: Stimulus control flow window graphically depicts the module processing flow of the stimulus

Multiple arrows may exit or enter a module. Multiple arrows exiting a module indicate either a split or a fork. A fork denotes a choice among the multiple paths. A split indicates all paths can be taken; that is; there is the potential for parallel processing of the modules to which control of the stimulus has been passed. Realization of this parallelism depends on the allocation of modules to tasks and tasks to processors. PEDESTAL supports conditional specification for passing control of the stimulus; these conditions can either be probabilistic or can be a function of the state of the simulated system. When multiple arrows enter a module, the user can specify whether an input from any of the arrows triggers invocation or whether inputs from all arrows are required to trigger the module. PEDESTAL's drawing package enhances the icons to show the results of design specification. For example, in Figure 4.1-3, M2 has been enhanced to show a split, M6 shows a join, and the arrow from M3 to M7 shows an intertask flow of control.

Module parameters consist of two parts: code and run-time (instance-specific) parameters. The module code is represented as a set of resource demand statements which include execute, transmit, lock, unlock, allocate, deallocate, and delay. Although the same named module can appear in multiple SCFs, its set of resource demand statements is essentially invariant across all instances of the module. However, such run-time parameters as priority and repetition count can vary by module instance. PEDESTAL supports both parameter types by allowing run-time parameter specification and a pointer to the code specification on the module instance. The set of resource demands can be specified either as a sequence of text commands or as parallel resources in a graphical resource demand window.

Task parameters that may be specified in PEDESTAL include priority, reentrancy, maximum instantiation count, and maximum buffer size.

The current version of PEDESTAL supports specification of operating system services (resource management) by allowing device contention selection and simple specification of instruction and data overheads caused by the operating system. The operating system specification is distributed over the various devices.

Data Architecture - The data architecture specification is provided by PEDESTAL's data stores which allow the user to set the size of the data store as well reflect the data store organization by specifying the number of physical accesses various logical I/O operations require.

Allocations - Once the hardware, software, and data architectures have been specified, the design specification is completed by allocating modules to tasks (functional partitioning), tasks to processors (functional allocation), devices to operating systems, and data stores to memory devices.

4.2 Translator

The translator is the keystone to PEDESTAL's flexible structure as it is designed to be extended or replaced with negligible impact on other language components. Thus, new concepts (e.g., reliability) or new applications (e.g., gates or circuits) require minimal changes to the PEDESTAL language in order to be modeled.

Most of the computer system specific knowledge is encapsulated in the translator. For example, the translator knows about tasks and generates the appropriate "get and frees" to task instances whenever task boundaries are crossed. It also conditionally generates resource demands based on the user specification of operating system overheads. It reads the data structures created by the visual interface and translates these into sequences of nodes that a transaction must visit as it is processed by the general purpose simulator. As an example of the intelligence embedded within the translator, Figure 4.2-1 shows the sequence of resource demands that can be generated from an intertask communication across multiple processors.

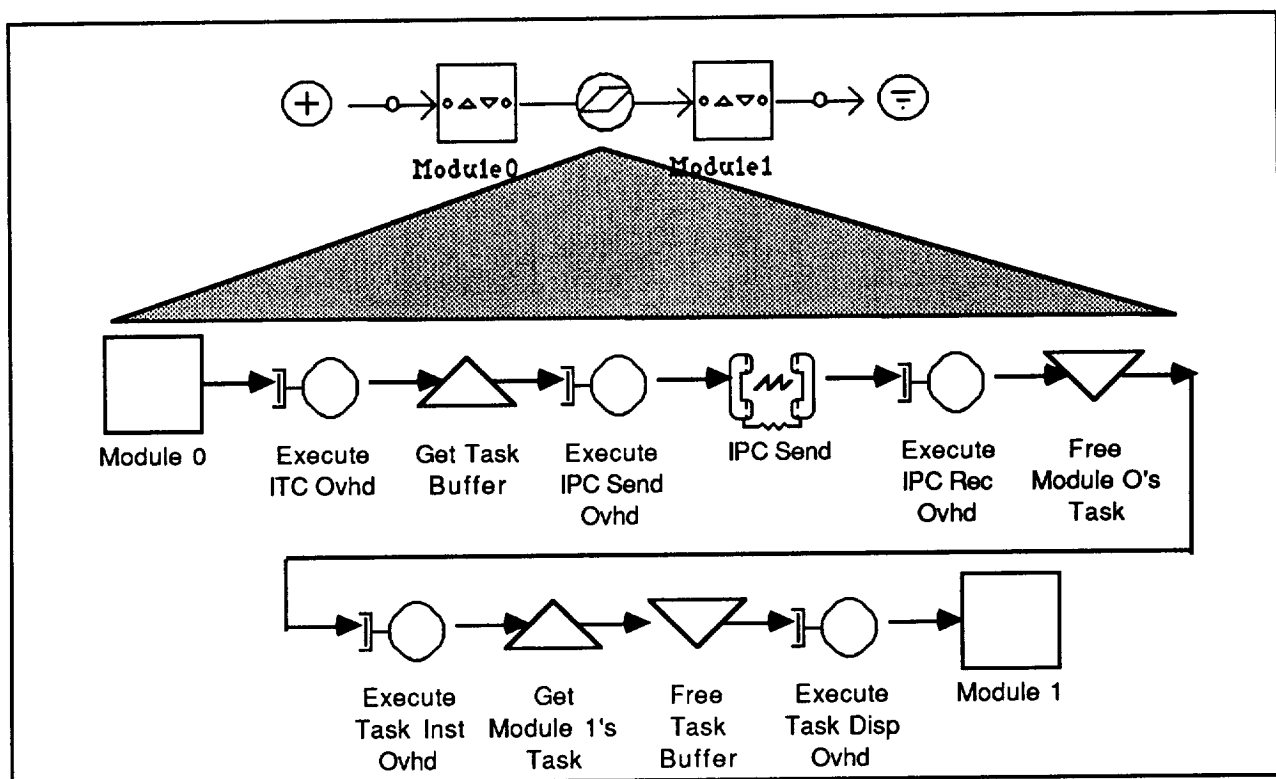


Figure 4.2-1: The translator contains computer system specific knowledge to generate operating system resource demands

4.3 Simulator

The simulator is a general purpose, process-based discrete-event simulation engine. The simulation engine is designed to permit experiments at any level of detail from gate-level to multisystem-level. However, as discussed above, it is the translator that presents the user with the system-level modeling constructs.

The simulator has been optimized for analysis of systems that have been specified in a top-down manner. Systems described as a hierarchy of increasingly detailed flows are directly translated into simulator primitives and executed. The simulator uses asynchronous timing (the most efficient method for systems which include a stochastic component) and can operate using either live (i.e., trace data) or theoretically generated arrival patterns. The PEDESTAL environment supports a multiprogramming environment in which the visual interface and the simulator share the host computer's program.

PEDESTAL's simulator data structures have been designed to support analysis using analytic techniques. Graph algorithms can be used to walk sequences of software nodes and compute the loads on the simulated devices.

4.4 Statistical Package

The statistical package collects, analyzes, and displays the results of model executions. The statistical package collects a large number of statistics automatically and can also collect statistics upon request by the user.

Automatic statistics collected include: device utilization, request queueing and service times at the devices, the number of service epochs at each device, module completion time, delay for task activation, and stimulus to terminator response time. In general, the statistics collected include means, standard deviations, minimums, and maximums.

Output data from the statistics package are displayed below the appropriate icons on the active window (under user control) and are written to disk files specified by the user. Both user-specified time sampled data and event data can be written to disk files. Finally, detailed system trace data can be selectively written to a PEDESTAL trace window. These detailed data provide a means to verify the behavior of the simulated system.

4.5 Software Interoperability

To exploit the interoperability of Macintosh software, interfaces to other software packages were designed. For example, figures from any of PEDESTAL's drawing windows can be cut or copied from PEDESTAL and pasted into a word processor or drawing package for preparation of a report.

To further illustrate the power of software interoperability, consider the following simple example. Figure 4.5-1 shows the SCF being analyzed. Data arrive from an external source at the rate of 40/second. The module ACK verifies the syntax and issues an acknowledgement to the sender; Local processes the data and issues an output message to a local user; Remote finishes processing the data and issues a message to a remote user. The three flags on the SCF indicate that response times are being collected at those flow points.

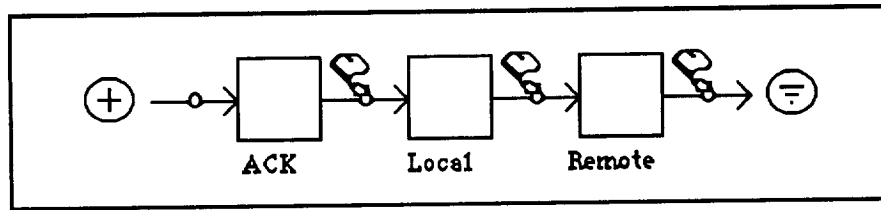


Figure 4.5-1: Stimulus Control Flow demonstrating multiple response time collection

The service and response time requirements of the three modules are shown below:

Service Times

ACK = 3 ms
Local = 6 ms
Remote = 6 ms

Response Time Requirements

ACK = 5 ms
Local = 50 ms
Remote = 100 ms

There is a single processor in the system. Intertask communication is via a buffer task and requires 3 ms. The design objective is to find a task partitioning that meets response time requirements. When relevant, ACK has highest priority followed by Local and then Remote.

Figure 4.5-2 shows the results from three different task partitionings. The three task

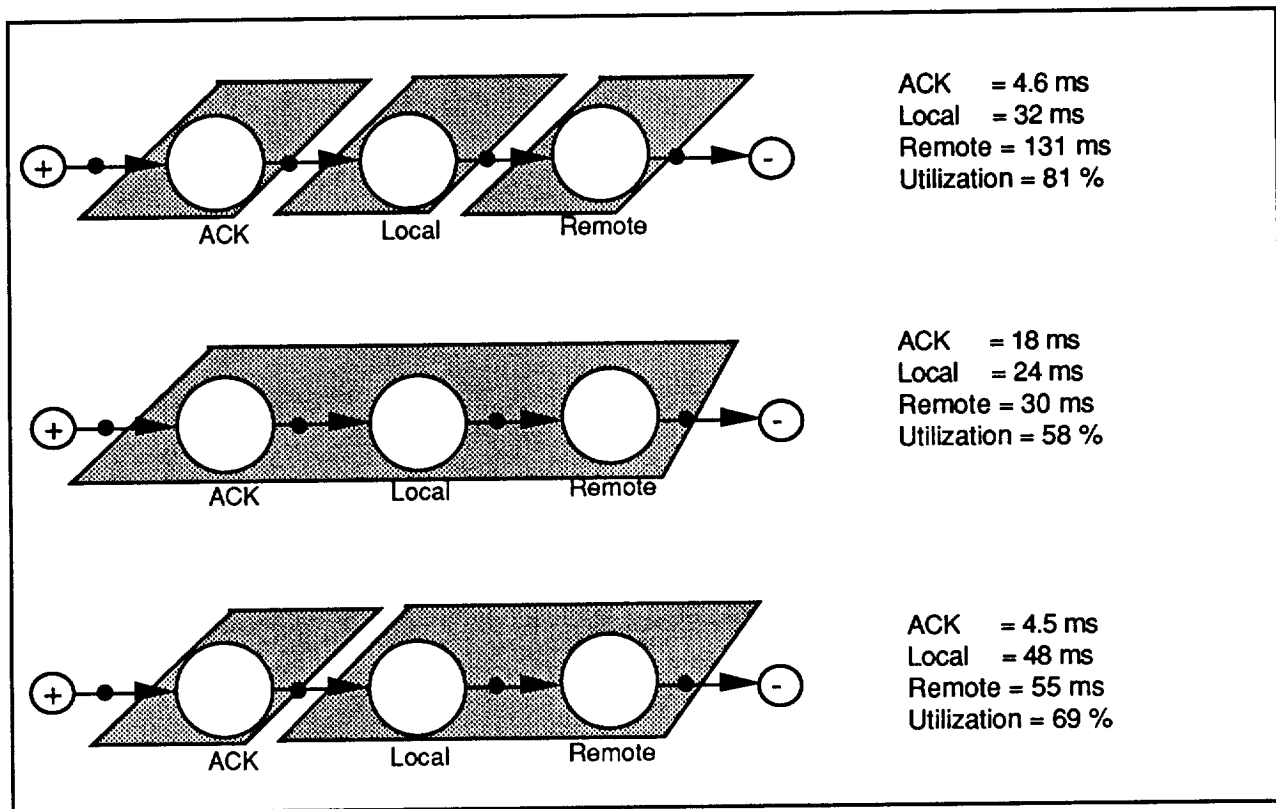


Figure 4.5-2: PEDESTAL results from three task partitionings

partitioning results in a high utilization due to the intertask communication overhead. Because of the high utilization and Remote's low priority, it fails to meet the response time requirement. Combining all modules in a single task results in a lower utilization, but because ACK response time includes all of the task queueing time, it fails to meet the 5 ms requirement. Putting ACK in its own task and combining Local and Remote in a second task results in all modules meeting their requirements.

Figures 4.5-3 shows the utility of the Macintosh software interoperability as it shows a summary graph produced by WingZ. Providing such a capability in PEDESTAL would not have been possible within the cost and schedule constraints.

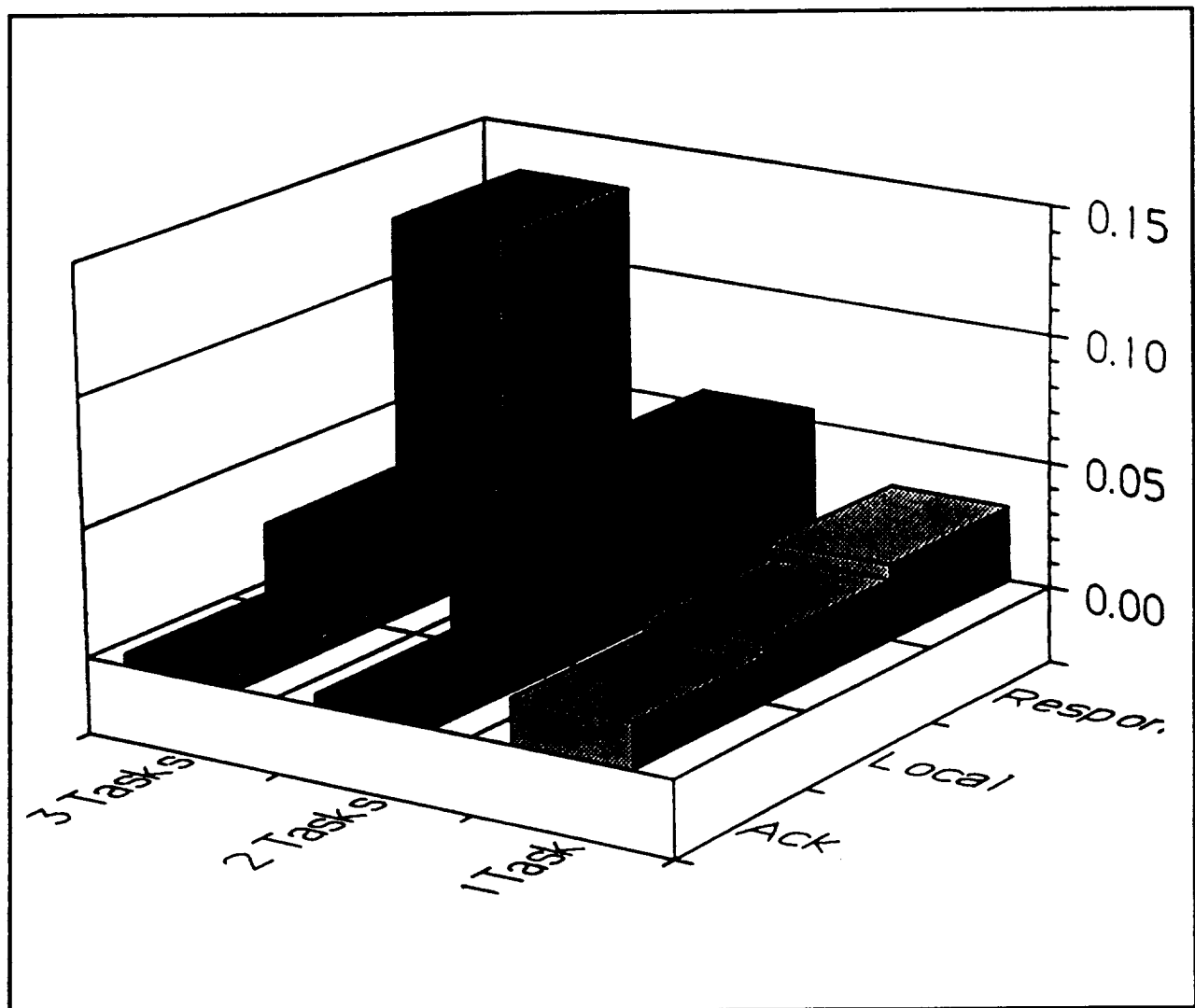


Figure 4.5-3: Summary graph from WingZ shows the utility of the Macintosh software interoperability

5.0 Results -- Sample PEDESTAL Models

During the course of PEDESTAL development and test, many sample systems were modeled. Detailed trace and statistical data were analyzed to verify and validate the prototype. Two of those sample systems are described in this chapter. This first sample provides simulation results to demonstrate validity; the second sample illustrates PEDESTAL's ease of specification.

5.1 Shared Resource Task Response Time Model

The following example is taken from the a paper written by Wesley Chu in the December 1988, "Real-time System Symposium." The purpose of the article is to investigate the effects of contention for shared resources on task response times. Figure 5.1-1 is taken from the above

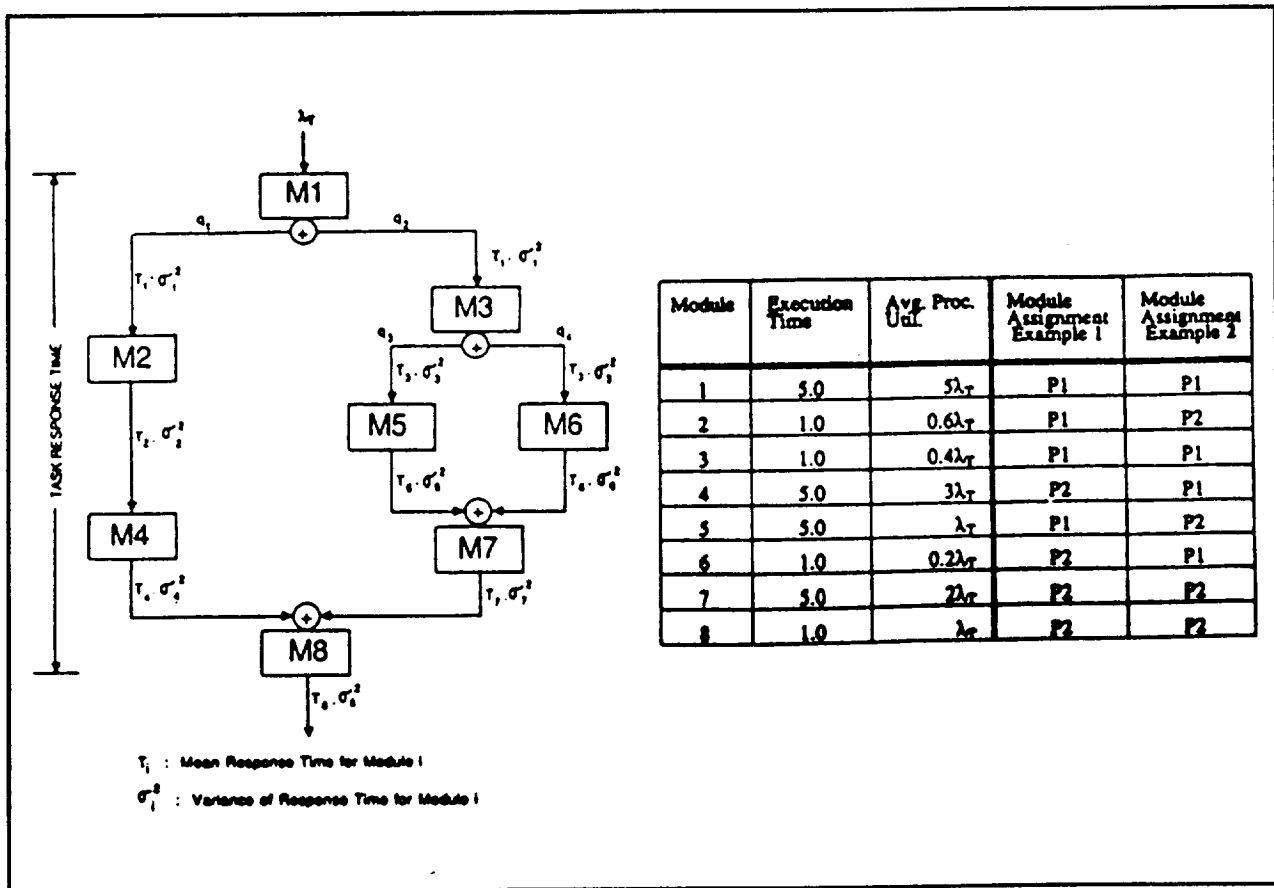


Figure 5.1-1: Sample SCF and processor demands

paper and shows the stimulus control flow and the module processor resource demands. Modules 1,2,4, and 8 require buffer acquisition before they can execute on their assigned processor. For module assignment 1, 3 buffers are allocated to P1 and 2 buffers are allocated to P2.

Figures 5.1-2 and 5.1-3 show the corresponding PEDESTAL SCF and set of resource demands for M1, respectively. Graphical sequences of resource demands have been entered for M1, M2, M4, and M8 as indicated by the enhanced icons for these modules. Note that these are the same modules that contend for the shared resources.

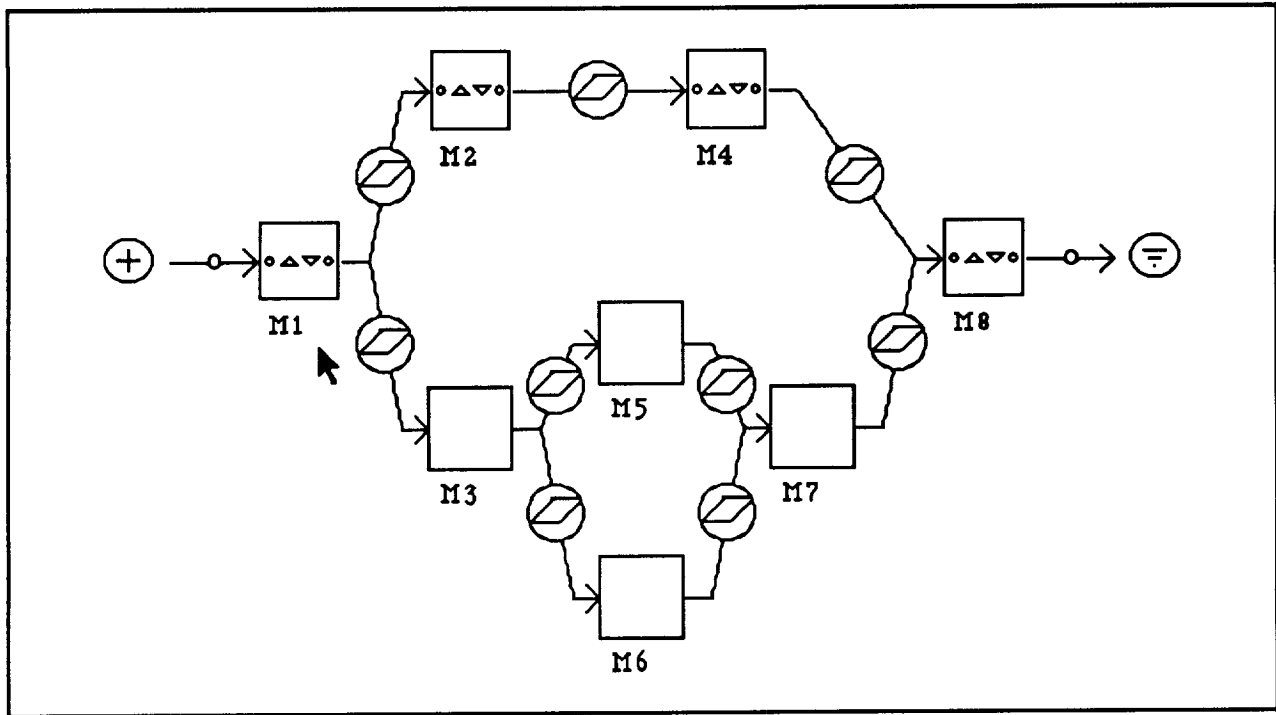


Figure 5.1-2: PEDESTAL SCF for the task response time example

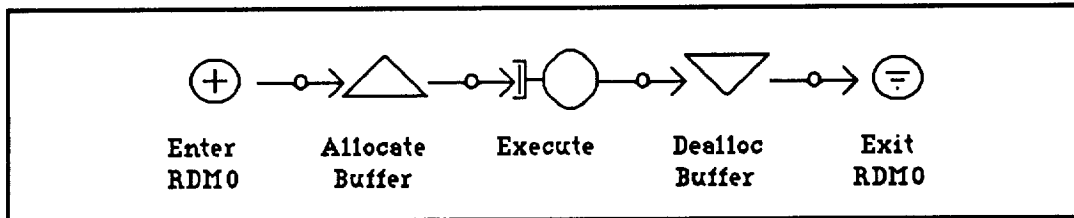


Figure 5.1-3: Sample sequence of resource demands for M1 shows acquiring and releasing shared resource

A comparison of results produced by PEDESTAL and interpreted from the graphs presented in Chu's paper follow. Since Chu did not present confidence intervals about the statistics, we have not reported any for PEDESTAL's values.

	<u>Stimulus arrival rate</u>	<u>Chu</u>	<u>PEDESTAL</u>
P1 Utilization	0.67 arrivals/sec	47%	49%
P2 Utilization		41%	41%
Response time (sec)		3.0	3.1
P1 Utilization	1.0 arrivals/sec	70%	72%
P2 Utilization		62%	59%
Response time (sec)		5.3	5.8
P1 Utilization	1.33 arrivals/sec	93%	94%
P2 Utilization		82%	72%
Response time		14.3	14.9

There is reasonably good agreement except for P2's utilization in the last experiment. There are many transactions in the buffer queues in this last run and as a result many transactions have not completed processing. Since M8 runs on P2 this may explain part of the discrepancy.

5.2 Telemetry Processing System

This example is based on the NASA Space Telescope telemetry processing system which was developed for the NASA Goddard Space Flight Center. Several modifications have been made to the system for the purpose of illustration. Figure 5.2-1 illustrates the major hardware components and connectivity.

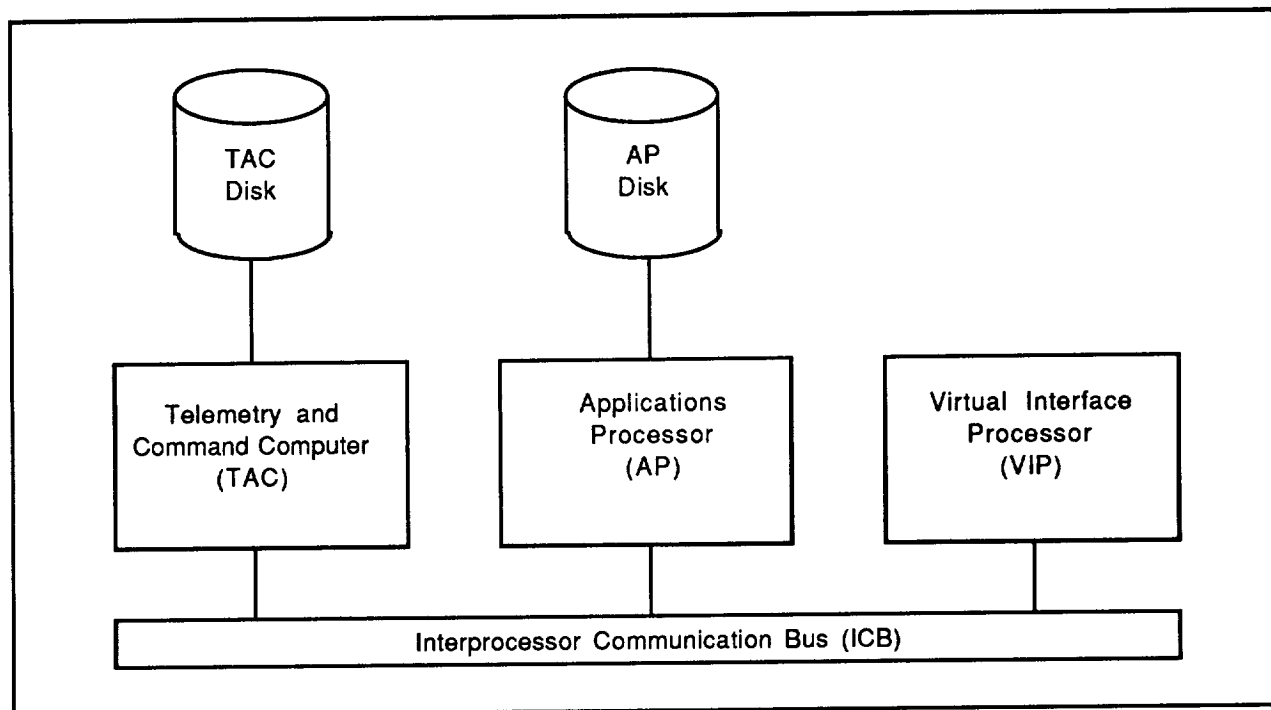


Figure 5.2-1: Major hardware components of the Telemetry Processing System

The system is comprised of three processors: Telemetry and Command (TAC) Computer, Applications Processor (AP), and a Virtual Interface Processor (VIP). These computers communicate over an Interprocessor Communication Bus (ICB). The TAC and AP computers are supported by peripheral disk drives.

During system operation, spacecraft telemetry data are received by the TAC which reduces the data into a form suitable for further processing. The reduced data are forwarded across the ICB to the AP. To maintain a complete record archive of all data received from the spacecraft, the TAC is responsible for capturing all incoming data on permanent storage media. Therefore, the TAC writes a copy of the raw telemetry data to a file on the TAC disk. At the same time, it prepares a backup copy which is sent across the ICB to the AP to be written to a file on the AP disk. This backup copy is maintained to ensure availability of archived telemetry data in the event that the primary copy on the TAC disk is corrupted.

Upon receipt of the reduced telemetry data from TAC, the AP performs some additional application processing and then forwards the processed data across the ICB to the VIP. The VIP

formats the data for display at a set of terminals where telemetry analysts are responsible for monitoring spacecraft operations. Based on the analysis of the incoming data, the telemetry analysts may wish to alter operational settings onboard the spacecraft. To perform this action, an analyst enters a command at a VIP terminal which is then sent by the VIP to the AP for processing. The AP then forwards the command to TAC for transmission to the spacecraft.

The representation of the Telemetry Processing System using PEDESTAL is described below.

Figure 5.2-2 shows the PEDESTAL hardware architecture window drawn to represent the sample system. Not shown on the diagram are devices connected to TAC which exchange data with the spacecraft by microwave link and the terminal cluster connected to the VIP.

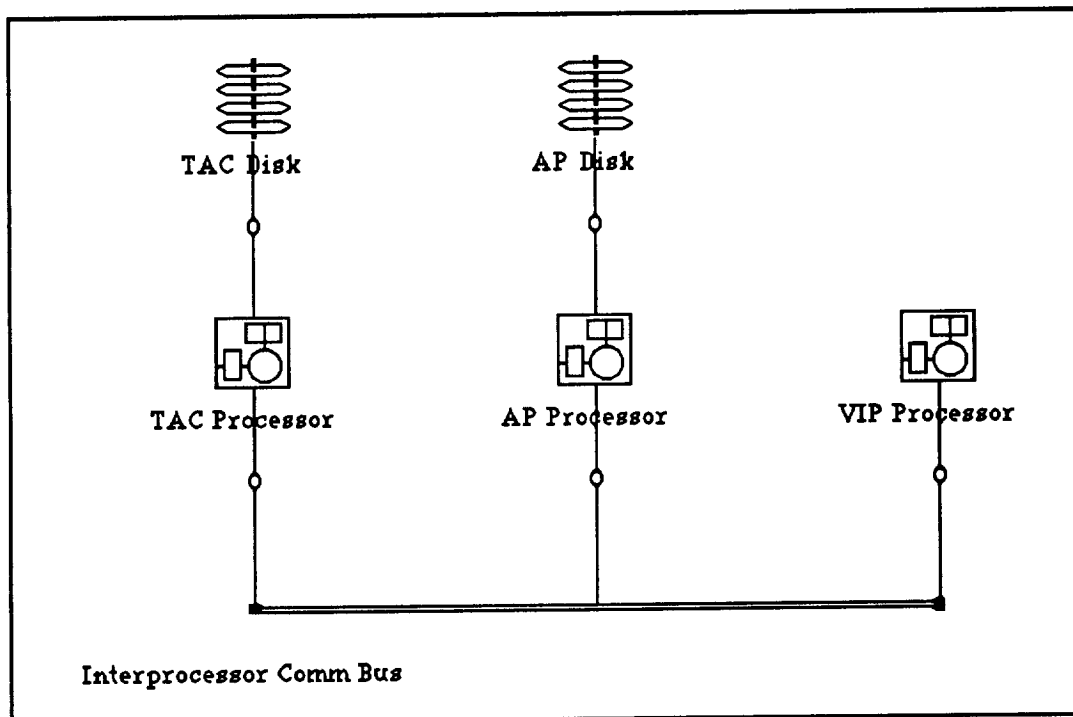


Figure 5.2-2: Telemetry hardware window from PEDESTAL

The flow of data and control in the system is captured by two PEDESTAL SCFs. The first SCF shown in Figure 5.2-3 includes that sequence of system modules that are triggered by the

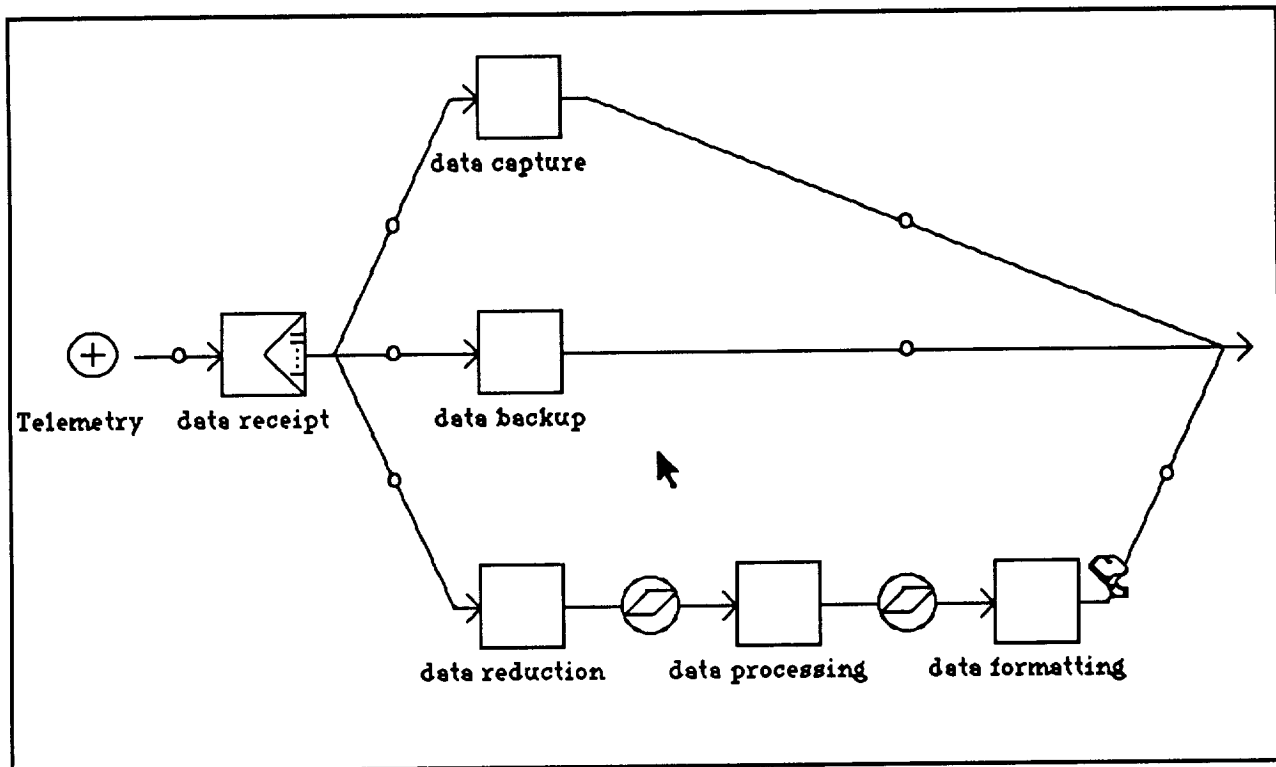


Figure 5.2-3: Telemetry data processing SCF shows potential parallel processing and response time delimiter

arrival of spacecraft telemetry data. The initiator node for the flow corresponds to the arrival of data to the system; each of the six modules included in the flow correspond to one of the primary telemetry processing functions described above.

The data capture, data backup, and data reduction modules are shown on multiple arrows departing the data receipt module. The enhanced icon denotes that the succeeding three modules can be performed in parallel. The response time flag indicates the end delimiter for a user-defined response time. Note that the response time does not include the data capture and data backup modules.

The second flow shown in Figure 5.2-4 contains three primary functions triggered by the entry of a command from a telemetry analyst. A response time requirement is also specified for this flow.

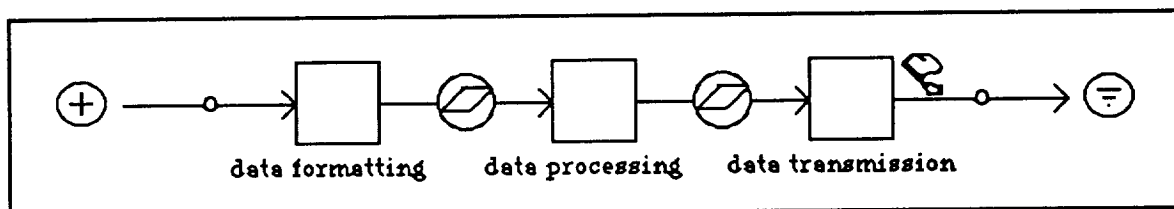


Figure 5.2-4: Command SCF shows task interfaces

Each module within an SCF is associated with a sequence of resource demands that are issued upon invocation of the module. For example, the data capture module in the telemetry SCF is

associated with the sequence of demands shown in Figure 5.2-5. The demands for this module include execution of 10,000 instructions and writing 20,000 bytes of data to a file called "tele data archive."

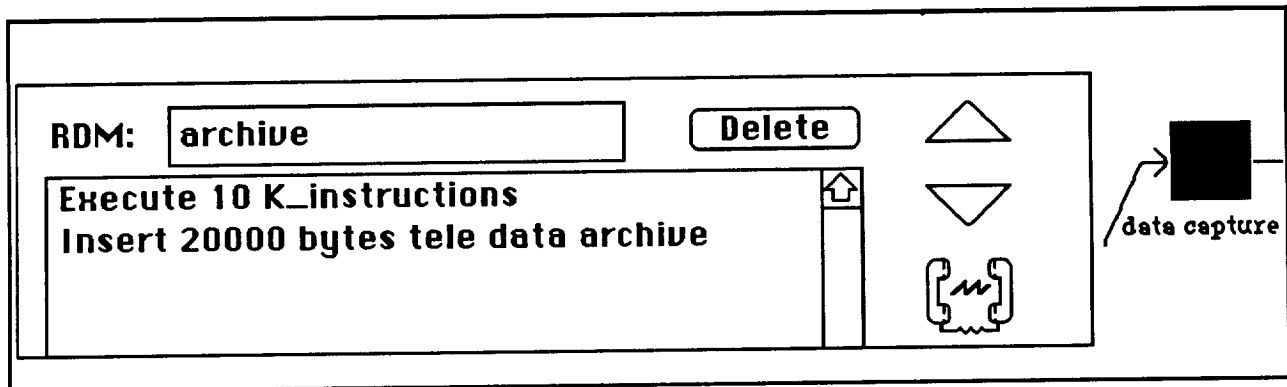


Figure 5.2-5: PEDESTAL sequence of resource demands for the data capture module

Some of the additional system data captured by PEDESTAL's visual interface are listed below:

- Workloads are specified in terms of data sizes and arrival rates associated with each of the two system stimuli. Telemetry data arrive once per second with a data size of 20,000 bytes and analyst commands are entered once per minute with a size of 1,000 bytes. Modifications to data sizes can be made by "opening" SCF arrows. For example, the data reduction module receives 20,000 bytes of input data but only outputs 5,000 bytes.
- Hardware performance characteristics such as processing speed and transmission rates are specified for each processor, memory, and bus.
- Data stores are identified by name in the resource demand which accesses them. The system has two data stores which hold raw telemetry data captured for archive purposes and backup copies of that same data, respectively.
- Operating system overheads are specified for two types of system services: interprocessor communication and communication with memory devices. These overheads are incurred whenever tasks on different processors communicate and whenever the data are sent to disks.

The final steps in the specification are allocation of modules to tasks, tasks to processors, and data stores to memory devices. In this system, each module is assigned to a separate task.

After a system has been described, the translator and simulator are invoked by issuing the RUN command. Sample device utilizations from a 500 second simulation are shown in Figure 5.2-6.

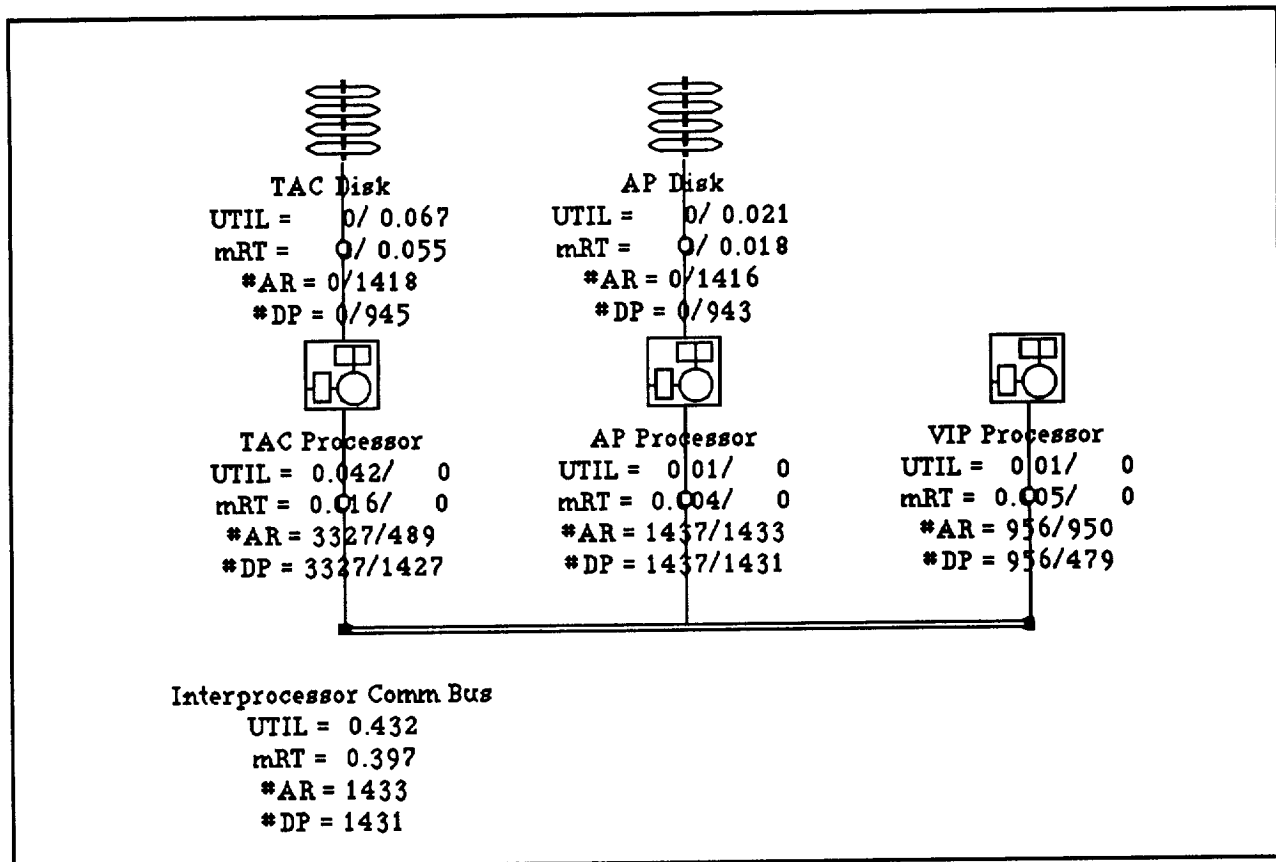


Figure 5.2-6: PEDESTAL output statistics show the bus to be the most highly utilized device

6.0 Conclusions and Recommendations

The major conclusion of this Phase II SBIR research and development effort is that complex, real-time computer systems can be specified in a non-procedural manner using combinations of icons, windows, menus, and dialogs. Such a specification technique provides an interface that system designers and architects find natural and easy to use. In addition, PEDESTAL's multi-view approach provides system engineers with the capability to perform the trade-offs necessary to produce a design that meets timing performance requirements. Sample system designs analyzed during the development effort showed that models could be constructed in a fraction of the time required by non-visual system design capture tools.

A second conclusion reached during the prototype development is that the Macintosh is a suitable platform for hosting PEDESTAL. The prototype was designed for the Macintosh II, but can execute on the Macintosh SE and Plus models as well. Of course, the complexity of the systems being analyzed drive the processing and memory requirements of the prototype. The prototype runs under the Multifinder operating system so other applications can share the processor during long simulation experiments.

Technical objective 1, which was to develop an evaluation version of a graphical, integrated modeling language, was satisfied during this Phase II effort.

A third significant conclusion reached during the development effort is that potential users of PEDESTAL desire to represent more design complexity than can be analyzed by straightforward analytic techniques. That is, users want to be able to specify such system features as

- complex branching conditions in stimulus control flows and resource demand sequences,
- complex routing conditions on hardware diagrams,
- load dependent resource management,
- transient workload arrival rates,
- complex flow fission and fusion conditions,
- nested acquisition of passive and active resources.

Although ad hoc analytic techniques are capable of analyzing certain combinations of these complexities (e.g., see "Task Allocation and Precedence Relations for Distributed Real-Time Systems," IEEE Transactions on Computers, June 1987 and "Analysis of the Fork-Join Queue," IEEE Transactions on Computers, February 1989), an automated approach necessary for translating designs into a form suitable for analysis by analytic techniques is not possible without substantially sacrificing the accuracy of the modeling results. Rather than produce analytic results that would soon be of little use due to their high degree of error, the development team opted to incorporate the ability to specify the design complexities suggested by the NASA-JPL staff and other potential users. The result is a tool that enables specification of more complicated designs than was originally envisioned; however, no analytic techniques suitable for analyzing these designs exist and therefore the prototype incorporates only a discrete-event simulator.

It is important to review the goals of the prototype set forth in the Phase II proposal which stated that a common system description capability would provide a means to drive both analytic and simulation techniques. However, to be feasible the descriptive constructs had to be fairly simple so that analytic solutions were possible. Furthermore, a stated goal was not to extend the state-of-the art in analytic modeling. Given that the system descriptive capabilities were extended beyond those capable of being analyzed by analytic techniques, the logical conclusion was to trade-off the increased power of the language with the ability to provide analytic solutions.

Technical objective 2, which was to determine the degree to which the language meets its objectives by evaluating ease of use, utility of two sets of performance predictions, and the power of the language constructs, was only partially satisfied.

During the development of the PEDESTAL evaluation prototype several lessons were learned that will prove beneficial in future tool commercialization and enhancement efforts.

Visual Interface Design Specification - Developing a visual interface for complex system design specification was more difficult than envisioned and probably explains why few such tools exist. Certainly there are many visual CASE tools, but almost none of these specify the temporal information that is necessary to analyze performance. It was specification of temporal information that was especially difficult in designing the visual interface. Multiple views, directed arrows, and priority fields provided the means to resolve temporal conflicts.

In addition, designing an interface that minimized user interactions and could operate on the small Macintosh SE screen was a challenge. This challenge was met by (1) providing multiple system views which reduced complexity and thus could be displayed on a small screen, (2) providing intelligent defaults (e.g., PEDESTAL has an automatic routing algorithm that requires no user inputs even for complex, interconnected networks), and (3) requiring the user to input information only once and then adding this information to pop up selection lists for future design choices.

Referentially Transparent Design Specification - Providing the user with the capability to independently specify hardware, application software, data, operating system, and task assignments was a difficult challenge to meet. The key to meeting this challenge was separating the interface data structures from the simulator data structures and developing the translator to convert one set to the other.

System Testing - The time and resources required to test the many paths through the prototype software was a task which was drastically underestimated. The approach we took in developing the prototype is one which we feel is justified in such a development effort. We attempted to show feasibility of as many concepts as were felt possible within the time and budget constraints. Demonstrating the capability on some limited test cases, we felt demonstrated feasibility of the concepts. Exhaustively testing all of the software paths would have severely limited the number of capabilities that could have been demonstrated in the evaluation prototype. Development of a commercial tool or even an operational prototype will require substantially more testing. It should be noted that during the later part of the test period, almost all software defects could be corrected in less than one hour.

Software Interoperability - Early in the development effort we had plans to develop fairly sophisticated statistical and plot software to assist the users in analyzing time-dependent data produced by the simulator. After attempting to develop a general plot capability, we realized that one of the advantages of the Macintosh platform is the interoperability of the software. We therefore opted to take advantage of the many millions of dollars expended on software

development and simply develop the means to port PEDESTAL data to existing word processors, drawing, statistical, and plot packages. Section 4.5 in this report demonstrates some of the benefits obtained from this approach. Future capabilities such as code and data management and documentation can easily be added to PEDESTAL by integrating with software products designed specifically to perform these functions.

The overall goal of the PEDESTAL research and development effort is to develop a performance engineering tool which can be applied to actual, complex computer system designs. The technology developed during Phase II should be transferred to system designers and architects so that the likelihood developed systems meet specified timing performance requirements is maximized. A first step towards transferring this technology to designers is additional test and evaluation of the PEDESTAL prototype. The prototype was developed to serve as a testbed for evaluation and enhancement of features required to support operational applications of visual interface performance prediction technology. Additional evaluation will ensure that a variety of application types (e.g., hard real-time, command and control, information system) and system features (e.g., different operating system and communication protocols, large scale systems, complex software control flows) are capable of being analyzed.

Experience with the prototype during Phase II revealed the primary enhancements required to make the prototype operational. These enhancements are required to elevate the PEDESTAL prototype implementation to the level of a commercial product. The recommendation of the Phase II effort is to pursue commercialization of the PEDESTAL prototype by performing the following:

Enhancements

Increase Design Specification Options - The prototype demonstrated the feasibility of allowing the user to specify (1) complex functions for determining flow branching conditions, service times, etc., (2) resource management (operating system services and communication protocols) policies, and (3) task interfaces. The number of options provided by PEDESTAL for it to be a commercial tool needs to be increased. Additional functions that return the state of the system are required. For example, functions are needed to return the remaining capacity of a memory or bounded queue, workload type, task priority, data element size, etc. IEEE 802.3 and IEEE 802.5 are examples of communication protocols that need to be added. Timed data buffering and synchronous message passing are examples of task interfaces that need to be more easily specified than the prototype currently supports.

Increase System Views - Additional system views are required to increase the degree of scalability of the prototype to large, complex systems and to present a more complete diagnosis of system performance. For example, the mapping menu provides a centralized, convenient means to change functional and data allocations. However, to view the set of allocations each processor and memory device needs to be examined. Although PEDESTAL supports hierarchical views of design objects (e.g., software and hardware folders), this view needs to be extended to display the various allocations.

The current set of performance statistics displayed on the screen provide a software view on the SCF windows and a hardware view on the hardware windows. Statistics that support the decomposition of the hardware utilizations into software components need to be added. Furthermore, although intermediate response time statistics can be logged and analyzed, display of these statistics needs to be more fully supported on the screen.

Static System Description Report - A capability to produce a hardcopy report of the system under study needs to be developed (other than dumping all of the screens to the printer). For

small systems this report is unnecessary, but for large systems such a report would seem to be a necessity.

User Extensibility - Regardless of how many options the tool provides, some users will also want a function (e.g., resource manager, branching condition, service time function, operating system demand) that is not provided. To support these users, a technique for allowing users to extend the language is required.

Undo and Cut and Paste - PEDESTAL supports some undo and cut and paste, but in developing large models a more complete capability is needed.

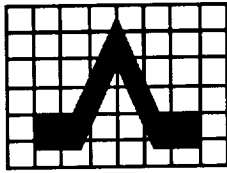
Testing and Documentation

More complete testing and documentation of PEDESTAL is needed before it could be considered for release as a commercial product.

APPENDIX

PEDESTAL

Preliminary User's Guide



ADVANCED SYSTEM TECHNOLOGIES, INC.

PEDESTAL™ Prototype Version 1.0

Designed & Developed by:
Duane Ball, Susan Hoyt, Oscar Steele, Gary Wright

Advanced System Technologies, Inc.
Copyright © 1989, Advanced System Technologies, Inc.

This work was sponsored by the U.S. Government under NASA Contract No. NAS7-995. This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7025 (APR 1984).

This data is furnished with SBIR rights under NASA Contract No. NAS7-995. For a period of 2 years after acceptance of all items to be delivered under this contract, the Government agrees to use this data for Government purposes only, and it shall not be disclosed outside the Government during such period without permission of the Contractor, except that, subject to the foregoing use and disclosure prohibitions, such data may be disclosed for use by support contractors. After the aforesaid 2-year period, the Government has a royalty-free license to use, and to authorize others use on its behalf, this data for Government purposes; but is relieved of all disclosure prohibitions and assumes no liability for unauthorized use of this data by third parties.

WARNING - Prototype Software

This software is a prototype designed to predict the performance of computer systems. Although several test cases have verified the accuracy of the performance measures, no claims are made concerning the accuracy of this tool.

TABLE OF CONTENTS

1. Introduction.....	1
2. Menus.....	3
2.1. Apple Menu.....	4
2.2. File Menu	5
2.3. Edit Menu.....	7
2.4. Draw Menu	9
2.5. Map Menu	14
2.6. Setup Menu	15
2.7. Run Menu	17
3. Expressions	18
3.1. Defining Expressions.....	19
3.2. Functions.....	21
3.2.1. User Defined Functions.....	22
3.2.2. Built-In Functions.....	24
4. Pedestal Models.....	64
4.1. Software.....	69
4.1.1. Software Control Flows.....	70
4.1.1.1. SCF & Macro Windows	73
4.1.1.2. The Stimulus/Terminator Icon	75
4.1.1.3. The Module Icon.....	80
4.1.1.4. The Delay Icon.....	85
4.1.1.5. The Note Icon.....	87
4.1.1.6. SCF/Macro Connectors.....	89
4.1.2. Resource Demand Modules	92
4.1.2.1. RDM Windows	94
4.1.2.2. Text RDMs.....	95
4.1.2.3. The Stimulator/Terminator Icon	96
4.1.2.4. The Allocate Icon	98
4.1.2.5. The Deallocate Icon.....	100
4.1.2.6. The Execute Icon.....	102
4.1.2.7. The Access Icon	104
4.1.2.8. The Request Lock Icon	106
4.1.2.9. The Relinquish Lock Icon	108
4.1.2.10. The Delay Icon.....	110
4.1.2.11. RDM Connectors	112
4.2. Tasks	114
4.2.1. The Task Window.....	115
4.2.2. The Task Icon	116
4.3. Locks.....	118
4.3.1. The Lock Window	120
4.3.2. The Lock Icon.....	121
4.4. Datastores.....	123
4.4.1. The Datastore Window	124
4.4.2. The Datastore Icon.....	125
4.5. Hardware.....	127
4.5.1. The Hardware & Node Windows	128
4.5.2. Hardware Devices	130
4.5.2.1. The Processor Icon.....	131
4.5.2.2. The Memory Icon	134
4.5.2.3. The Bus Icon	137
4.5.3. The Node Icon.....	140
4.5.4. The Boundary Icon.....	142

4.5.5. Hardware Connectors.....	144
4.6. Manager Window	146
4.6.1. The Manager Icon	147
4.7. The Desktop Window.....	150
4.7.1. The Task Window Icon.....	152
4.7.2. The Datastore Window Icon	153
4.7.3. The Lock Window Icon	154
4.7.4. The Manager Window Icon.....	155
4.7.5. The Software Sideview Window Icon.....	156
4.7.6. The Hardware Sideview Window Icon	157
4.8. Sideview Windows.....	158
5. Association Lists.....	160
5.1. Association List Dialog.....	161
5.2. Module -> Task	163
5.3. Task -> Processor	165
5.4. Datastore -> Memory	167
5.5. Device -> Manager	169
6. Statistics.....	171
6.1. On Screen Statistics.....	172
6.1.1. Icon Statistics.....	173
6.1.2. Connector Statistics	177
6.2. File Statistics.....	178
6.2.1. Periodic Statistics.....	179
6.2.2. Event Statistics.....	181
6.2.3. Importing File Statistics into Other Applications	183
6.2.3.1. Excell™.....	184
6.2.3.2. Exstatics™.....	186
7. Running a Model.....	188
7.1. Mapping Requirements	189
7.2. Software Window Requirements.....	190
7.3. Hardware Window Requirements	191

1. INTRODUCTION

This document describes Pedestal™, a simulation system developed by Advanced System Technologies, Inc.. Pedestal™ provides a graphical, common system description language which captures all of the essential performance data required to effectively do performance engineering analyses of complex, real-time computer systems.

This manual assumes familiarity with the Apple Macintosh™ family of computers and the elements of the standard Macintosh interface. The user who is unfamiliar with the Macintosh interface should read the manual supplied with the Macintosh or any introductory book which explains Macintosh operation.

Application of computer performance engineering (CPE) techniques throughout the design and development of complex, real-time computer systems is absolutely essential for ensuring that the delivered systems meet their specified performance requirements. Unfortunately, this specialized engineering is frequently not applied or the available performance prediction techniques are misapplied during system development and as a result systems fail to perform during system testing. Costly and time-consuming redesign efforts must then be initiated to correct the systems' performance deficiencies.

The reasons for the lack and misapplication of performance techniques are twofold: (1) the techniques require a specialized knowledge, which is in short supply, and (2) each of the two primary techniques available for predicting performance (analytic and discrete-event simulation) has deficiencies that can cause the analysis to fail if not applied by an experienced performance engineer.

Pedestal was developed to provide a tool which can help overcome these shortfalls in applying CPE techniques. By providing an integrated tool that can easily be used by a system designer as well as a performance engineer, and delivering a simulation engine which was designed by experts in discrete-event simulation and is thoroughly tested, Pedestal™ can make CPE an easier task.

The advantages of Pedestal™ are numerous. First, Pedestal™ has been specifically tailored to describe computer systems. This specificity is manifest in both the user interface of the tool as well as in the fundamental design of the simulation engine. The specialized interface provides those responsible for designing systems with a tool which can be easily used during all phases of design and development to assess performance impacts of various decisions. This is not to suggest that all design decisions should necessarily be based on performance, but at a minimum the performance impact of a decision will be available. The performance impacts will help to steer designers away from alternatives that may be attractive from a software engineering standpoint (e.g., extremely modular and highly isolated software), but can be disastrous from a performance perspective.

Secondly, Pedestals' graphic design language is used to describe the salient characteristics of system workloads, software behavior, and hardware characteristics. This design is then translated to a form more appropriate to the simulation engine. This approach allows easy expansion of Pedestal to include performance evaluation techniques such as analytics. All that is required is a new translator and an evaluation engine for the new technique (one of AST's future goals for Pedestal is to incorporate analytics). When such an addition is made to the capabilities of Pedestal, no changes need be made in the design language. Additionally, previously created models will automatically be able to use the new techniques.

2. MENUS

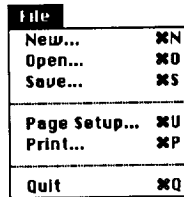
Those readers unfamiliar with standard operation of Macintosh™ menus should refer to the documentation provided with their computer or to one of the many excellent books available on this subject. Pedestal™ menus operate in the same manner as the standard menus with selection by mouse and in some cases by a keyboard command indicated on the menu beside the item. This Section addresses the Pedestal™ menus in the order in which they appear left to right when the Pedestal™ application is running.

2.1. APPLE MENU

The standard apple menu appears in all Macintosh™ applications. This menu provides a method of accessing desk accessories (and other running applications if Multifinder is being used). Traditionally, the first one or two menu items in the apple menu provide information about the application. Pedestal™ adds an item called "About Pedestal™..." at the top of the apple menu. This item will display the Pedestal™ help dialog.

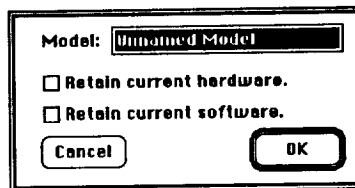
2.2. FILE MENU

The file menu, shown in Figure 2.1, provides the standard file manipulation commands. All of the windows, mappings, and statistics definitions (see Chapters 4, 5 and 6) which comprise a Pedestal™ model are kept in a single file, one for each model. The file I/O items in the file menu work with these files. The print items, however, affect only the currently active (frontmost) window of a model.



The File Menu
Figure 2.1

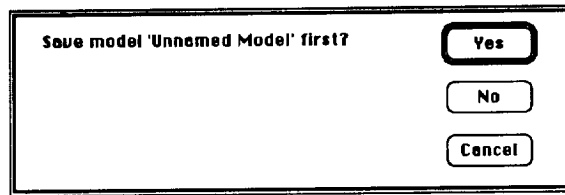
The "New..." item is used to create a new model file. When this item is selected the new model dialog, shown in Figure 2.2, is displayed. This dialog allows the specification of the new model's name and provides two check boxes enabling the user to retain a copy of the software and/or the hardware of the currently loaded model.



The New Model Dialog
Figure 2.2

The "Open..." item displays the Macintosh™ standard file dialog to allow the user to choose the file to be opened. The "Save..." item also displays the Macintosh™ standard file dialog, this time allowing the model file to be saved in any folder by the user's chosen name. Next in the menu is the "Page Setup..." item; this item calls up the standard page setup dialog to allow modification of the way printing is performed. "Print..." will print the contents of the model window which is currently active. All windows in Pedestal™ may be printed. The last item in the file menu is "Quit"; this will terminate the application.

Since the new, open and quit commands could cause the loss of information added to the model currently open, each of these commands will check to see if the current model has been modified since the last time it was saved. If so, the dialog in Figure 2.3 will be displayed to allow the current model to be saved before the menu command is executed.



The "Save First?" Dialog
Figure 2.3

2.3. EDIT MENU

The edit menu, shown in Figure 2.4, provides the following standard editing functions with keyboard equivalents: cut, copy, paste, select all, and duplicate.

Edit	
Get Info	⌘I
Cut	⌘H
Copy	⌘C
Paste	⌘V
Select All	⌘A
Duplicate	⌘D
Functions	⌘F
✓Full Warnings	

The Edit Menu
Figure 2.4

Pedestal™ windows (see Chapter 4) contain drawings composed of icons and, for some windows, lines (called connectors) connecting the icons. Often there is more than one data structure associated with one connector/icon. The primary data structure is always accessed by double clicking on the connector/icon. Selecting a connector/icon by clicking on it once, then selecting the "Get Info" will access the secondary data structure associated with the connector/icon.

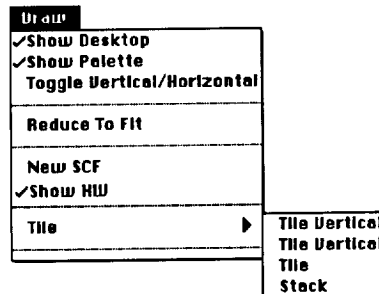
The next five items in the menu are standard edit commands for most Macintosh™ programs. The "Cut" command removes the currently selected icons and/or connectors in the active window and makes copies of them in the Clipboard (a sort of temporary memory) after removing the previous contents of the Clipboard. Since connectors must have an icon on each end, cutting an icon can cause connectors which were not selected to be deleted. These connectors will not be put in the Clipboard. Likewise, connectors which are selected and have one or both of the icons on its ends unselected will be deleted but will not be put in the Clipboard. Additionally, some icons can not be cut and will be ignored even if they are selected. The "Copy" command works just like the "Cut" command with the exception that nothing on the active window will be deleted.

The "Paste" command will change the cursor from the "pointer" to the "place" version. Clicking on the window will then cause the contents of the Clipboard to be copied into the window. If the type of window the Clipboard contents were copied from is different from the window type you attempt to paste it into, the paste command will not work. The "Select All" command provides a simple way to select all the icons and connectors in the active window. The last standard edit command is "Duplicate". Choosing this item is the same as choosing "Copy" and then immediately choosing "Paste".

The "Functions" item of the edit menu displays the function definition dialog explained in Chapter 3. The last item in the menu is a flag. When "Full Warnings" is checked, all incorrect or invalid actions will cause a dialog to be displayed which will explain the problem. If the flag is not checked, these same actions will only cause an audible beep to sound. By default, this item is checked.

2.4. DRAW MENU

The draw menu provides commands which pertain to the various windows in Pedestal™. The draw menu is shown in Figure 2.5.

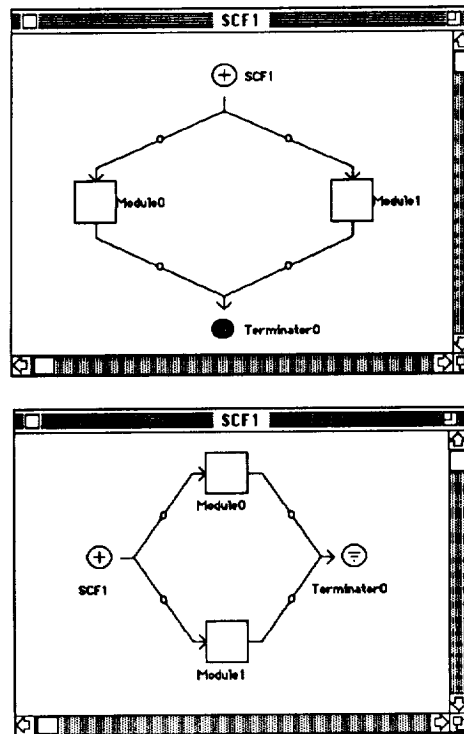


The Draw Menu
Figure 2.5

The desktop window (see Section 4.7), contains icons for every window defined in the current model. These windows may be accessed by double clicking on the appropriate icon. Sometimes, if the Macintosh™ desktop is getting too crowded, the user may not want to have the desktop window visible. The "Show Desktop" flag on the draw menu controls the visibility of the desktop window. If checked, the desktop window will be visible. If not checked, the desktop window will be hidden. By default, this item is checked.

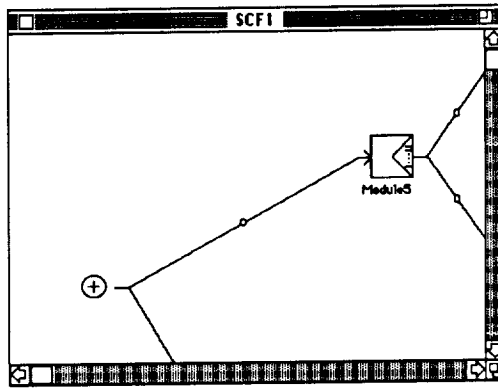
For similar reasons, the user may not want and/or need the palette (see Chapter 4) to be displayed. The "Show Palette" flag on the draw menu allows control of palette visibility. If checked, the palette will be displayed. If not checked, the palette will be hidden. By default, this item is checked.

"Toggle Vertical/Horizontal" is the next item in the draw menu. Software window connectors (see Chapter 4) are directed. That is they are drawn with arrows on one end. Certain rules are used in determining how to draw these connectors based on the positional relationship of the two icons the connector connects. The rules must make an assumption as to the normal direction of flow in order to work well. The "Toggle Vertical/Horizontal" command switches the assumption from vertical to horizontal or vice versa. Vertical mode will produce the best drawings if the majority of connections are made from top to bottom in the window. Horizontal mode will produce the best drawings if the majority of connections are made from left to right. Figure 2.6 shows two windows, the first is an example of the vertical mode, the second is an example of the horizontal mode. Note that changing modes only changes how the connectors are drawn, it does not reposition any icons. If required, icons must be repositioned manually.

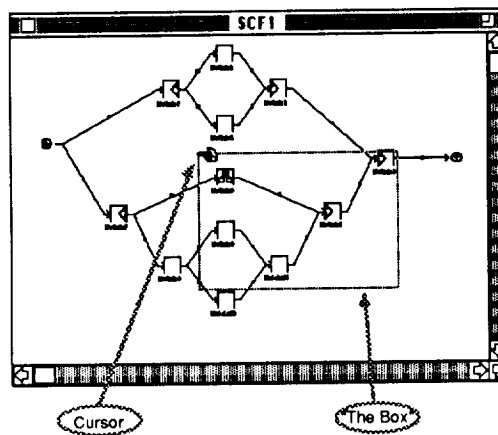


Examples of the Vertical and Horizontal Drawing Modes
Figure 2.6

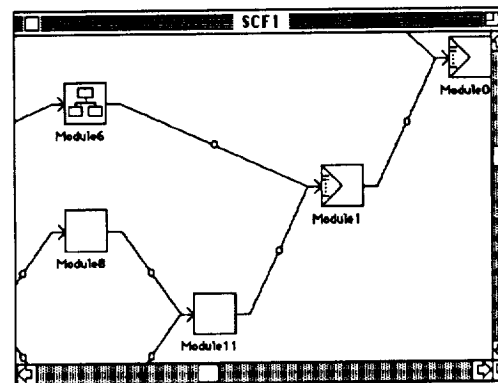
All drawing windows in Pedestal™ (see Chapter 4) are scrollable, thus allowing the size of the drawing to exceed the size of its window. The scroll bars on a window allow different parts of the diagram to be shown. However, for very large drawings, scrolling to certain position may be tedious. The "Reduce To Fit" command allows a more convenient method of choosing which part of a large drawing is to be shown in the window. If the active window is a drawing window, selecting this command will display a the entire drawing in the window, shrinking the drawing to the necessary size. The user can then pick the part of the drawing to be displayed and the drawing will zoom back to normal size and scroll automatically (in one jump) to the chosen portion of the drawing. Figure 2.7 shows a window whose drawing is larger than the window. Figure 2.8 shows what the user will see after selecting "Reduce To Fit". The gray box is sized such that it represents the size of the window if it was shrunk the same amount as the drawing. The box is tied to the cursor and will move with the cursor, though it will not move outside the window. After positioning the box over the part of the drawing the user wishes to appear in the window, clicking the mouse will cause the drawing to return to normal size and the window will display the part of the drawing which was in the box. Figure 2.9 shows the window after the mouse is clicked with the box in the position shown in Figure 2.8.



The Window Before "Reduce To Fit" is Selected
Figure 2.7



The Window After "Reduce To Fit" is Selected
Figure 2.8

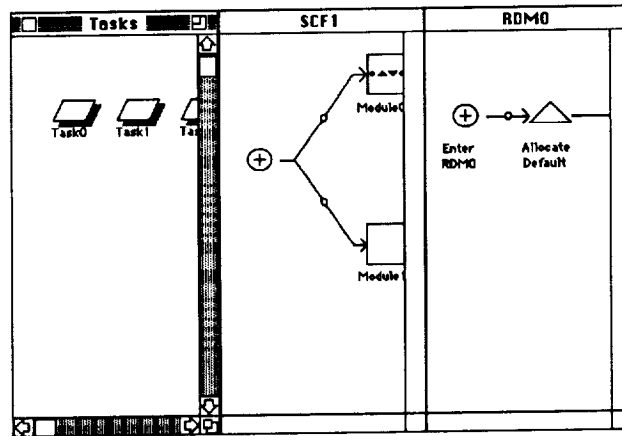


The Window After The Mouse is Clicked
Figure 2.9

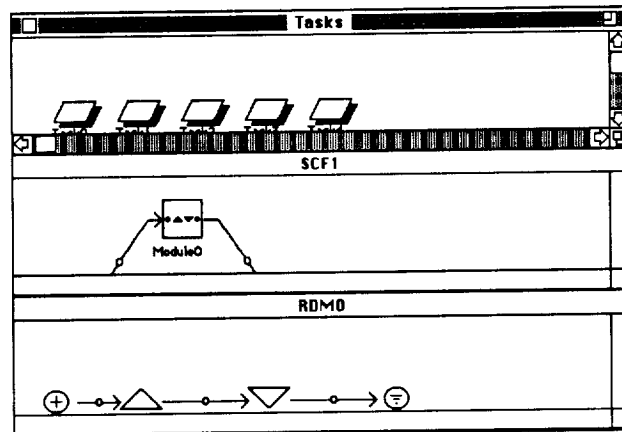
The "New SCF" item in the draw menu creates a new SCF window (see Section 4.1.1.1). This window will have a name distinct from all existing window names in the model. There is only one (top level) hardware window (see

Section 4.5.1). The "Show HW" item will make the hardware window visible, if it is not already, and will bring that window to the front of all other open windows.

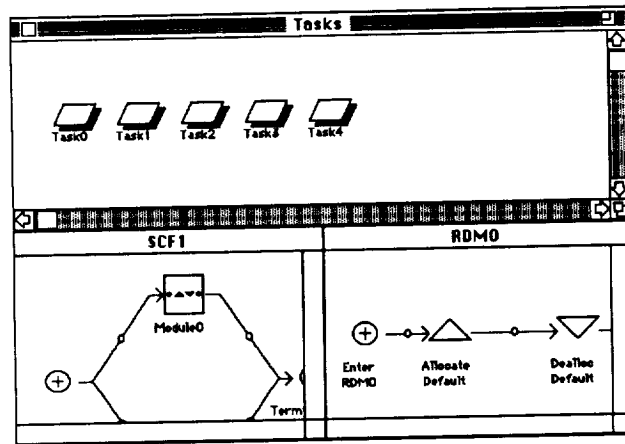
The "Tile" item in the draw menu brings up a submenu. The items in the submenu will cause all open windows (except the desktop window) to be resized and moved such that no window is totally obscured. Figures 2.10, 2.11, 2.12 and 2.13 show examples of the effects of the "Tile Vertical", "Tile Horizontal", "Tile" and "Stack" commands on the Macintosh™ screen.



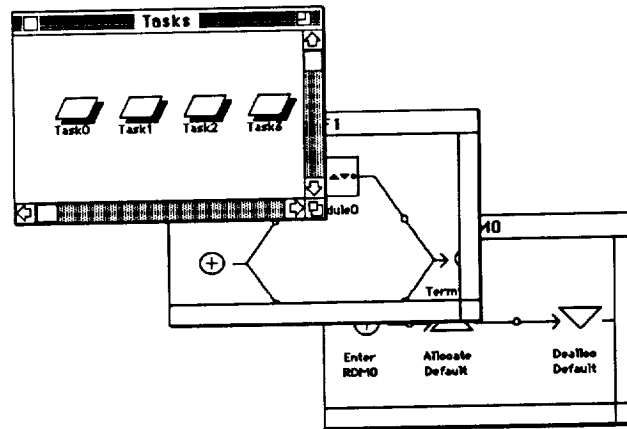
Vertically "Tiled" Windows
Figure 2.10



Horizontally "Tiled" Windows
Figure 2.11



"Tiled" Windows
Figure 2.12

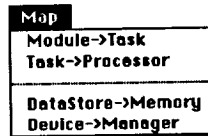


"Stacked" Windows
Figure 2.13

The draw menu is also used to provide an easy way to bring any open window to the front. The names of all open windows are appended to the draw menu. At any time, selecting one of these window names will bring the associated window to the front of all other windows.

2.5. MAP MENU

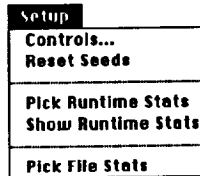
The map menu, shown in Figure 2.14, provides commands which allow the various mappings to be viewed and/or edited. The Mapping concept and the purpose of the mappings is discussed in detail in Chapter 5.



The Map Menu
Figure 2.14

2.6. SETUP MENU

The setup menu, shown in Figure 2.15, provides the commands necessary to control the environment of a simulation.

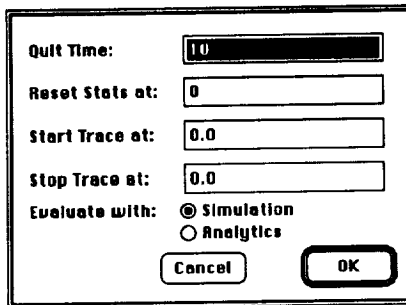


The Setup Menu
Figure 2.15

The "Controls..." item displays the simulation control dialog shown in Figure 2.16. The two radio buttons "Simulation" and "Analytics" are intended to allow the user to specify which model evaluation method is to be used: discrete event simulation or analytical approximation. At the present time, only simulation is supported. The "Quit Time" field specifies the amount of time to be simulated. The "Reset Stats" field specifies when stats collection should be restarted. This is used so that the effect of start up transients can be removed from the statistics reports if an estimate of the transient time is known. For low level analysis Pedestal™ can provide trace files which trace the paths of transaction through the simulation engine. Simulation time is recorded with information about transaction type, task, transaction identification and messages by which the executing portion of code can be identified. The "Start Trace" and "Stop Trace" fields specify what portion of the simulation the trace file will cover. Trace files are very useful for the user familiar with the internal workings of the simulation engine. The trace is voluminous and greatly reduces simulation speed. It is suggested that the trace only be generated if it is needed. The trace may be viewed during the simulation by double clicking the trace icon on the desktop window. Since a large volume of information will be produced, the trace will be written to a file, and only the last 32K of text will be visible in the trace window.

The "Reset Seeds" item on the Setup menu causes all random number streams to be set to their initial values so that models can be run in identical environments with repeatable results. The "Pick Runtime Stats" item allows specification of which statistics will be displayed and "Show Runtime Stats" is a menu check item that when set causes the chosen statistics to be displayed during run time. By default, "Show Runtime Stats" is checked. More about stats including the method of picking the stats to be displayed is presented in Chapter 6. Pick file stats allows the user to choose stats to be written to a file.

Note that all times on the control dialog are in seconds.



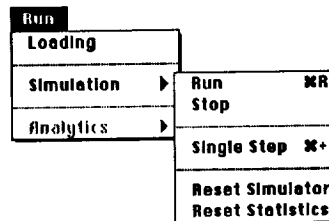
The Simulation Control Dialog is a rectangular window with a thin border. It contains several controls arranged vertically on the left side, each followed by a text input field. The controls are: 'Quit Time:' with a field containing '10'; 'Reset State at:' with a field containing '0'; 'Start Trace at:' with a field containing '0.0'; 'Stop Trace at:' with a field containing '0.0'; and 'Evaluate with:' followed by two radio buttons. The first radio button is labeled 'Simulation' and is selected (indicated by a filled circle). The second radio button is labeled 'Analytics' and is unselected (indicated by an empty circle). At the bottom of the dialog are two buttons: 'Cancel' on the left and 'OK' on the right. The 'OK' button is slightly larger and has a double border.

Quit Time:	10
Reset State at:	0
Start Trace at:	0.0
Stop Trace at:	0.0
Evaluate with:	<input checked="" type="radio"/> Simulation <input type="radio"/> Analytics
<input type="button" value="Cancel"/> <input type="button" value="OK"/>	

The Simulation Control Dialog
Figure 2.16

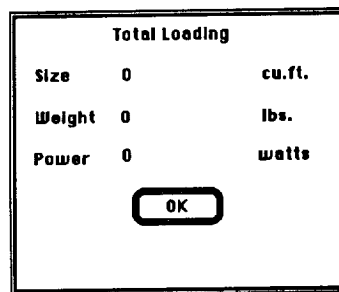
2.7. RUN MENU

The run menu, shown in Figure 2.17, is used to control the simulation of a model.



The Run Menu
Figure 2.17

The first item in the run menu is "Loading", choosing this item will display the total loading dialog shown in Figure 2.18. Hardware components have as part of their specification physical characteristics; the sum total of the characteristics for all devices in the model is shown here.



The Total Loading Dialog
Figure 2.18

The "Simulation" item of the run menu provides a submenu. The "Run" command will cause the current model to be translated and simulated. For an explanation of the process of running a model (or producing a runnable model) see Chapter 7. The "Stop" command will temporarily pause the simulation. When a simulation is paused, the "Single Step" command can be used. Single stepping will resume the simulation for the time necessary to process one event then will again pause the simulation. When paused, the "Reset Simulator" command can be used. Resetting the simulator cancels the simulation run entirely. Additionally, when the simulation has paused, the "Run" command is renamed as "Continue" and selecting this item will continue the simulation run, taking it out of the single step mode. The remaining command "Reset Statistics" is used to restart the collection of statistics at the current point in the simulation.

3. EXPRESSIONS

Pedestal™ contains many dialogs. These dialogs are used to define the various objects in Pedestal™. A large number of fields in these dialogs are number fields. To increase its expressive power, Pedestal™ allows any number fields to be entered as an expression.

Expressions consist of numbers, function calls, and the standard mathematical operators: +, -, *, /. The functions which can appear in an expression are chosen from a list of supplied functions and user defined functions. The arguments to these functions may also be expressions.

All expressions and all numbers within expressions are treated as real numbers. For the few fields that require an integer value or a value within a specific range, the value of the expression will be truncated and/or reduced/increased as needed to transform it to proper form. Note that this will occur *after* the expression is evaluated.

Some fields are evaluated when a simulation starts, others are evaluated repeatedly as a simulation progresses. For example, the time interval between work requests being generated by a workload is re-evaluated after each request so that the interval is not necessarily constant. Conversely, the execution rate of a processor is evaluated once at the start of the simulation; and is fixed throughout a simulation run.

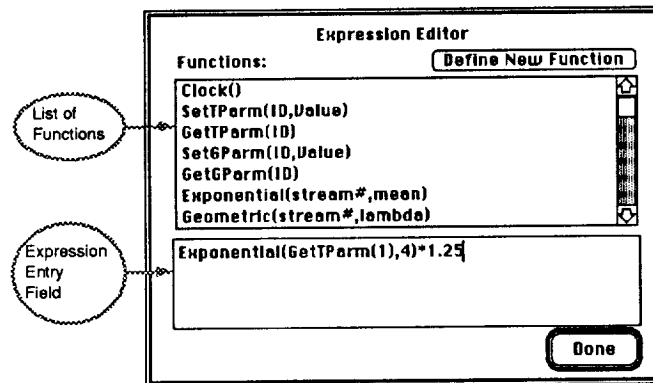
Functions whose results can vary with time are called time varying functions. Most of the built-in functions supplied by Pedestal™ are time varying. Any user defined function which references a time varying function is also considered to be a time varying function. Whether a field is evaluated once or many times during a simulation determines which functions may be used in an expression entered in the field. A field which is only evaluated once at the start of the simulation can not have any references to a time varying function.

3.1. DEFINING EXPRESSIONS

Any number field of any dialog (except the print and page setup dialogs) can be entered as an expression. The expression may be entered by directly typing it in the field or by using the expression editor dialog. To use the expression editor dialog, hold down the command key (⌘) and double click on the field whose value is to be entered.

Expressions can contain numbers and function invocations and a few math symbols. The math symbols are +, -, *, and /. These symbols correspond to addition, subtraction, multiplication, and division. Unary + and - is also supported. Evaluation of an expression follows normal precedence rules. Parentheses may be used to change the order of evaluation. Numbers may be entered as integers or reals. Reals may be entered in scientific notation if desired. Note that for real values, a digit must precede the decimal point; i.e. .023 must be entered as 0.023.

The expression editor dialog, shown in Figure 3.1, contains a list of functions and a field for the expression to be entered in. The list of functions will contain all built-in and user defined functions which can be used in the field which was clicked in to bring up the dialog. There is also a "Define New Function" button which will display the function definition dialog (Section 3.2.1) to allow new user functions to be defined.



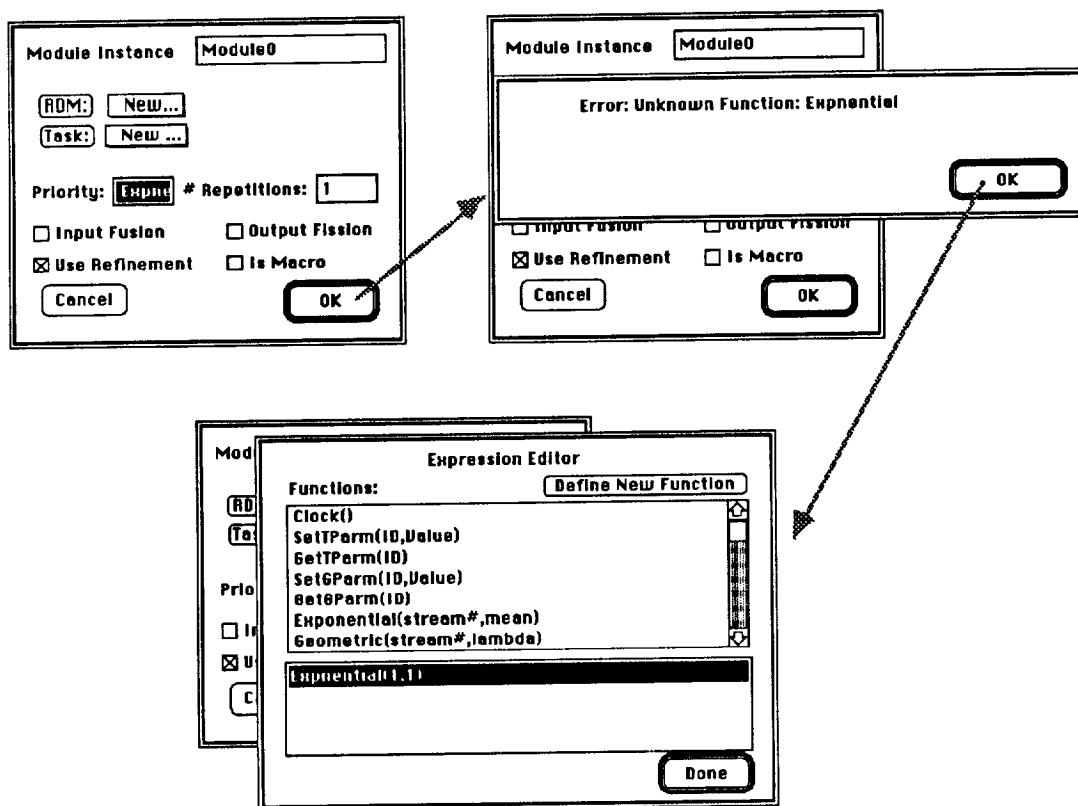
The Expression Editor Dialog
Figure 3.1

To enter an expression just type in the expression entry field. To include a function, either type its name or double click on its name in the function list. Be sure to replace the parameter names with values (or expressions).

When the "OK" button is clicked on a dialog, all number fields are checked to ensure that the expressions are syntactically correct. An expression entered in the expression editor dialog is also checked when you click on the "OK" button. If there is an error, a dialog will appear which will try to give you some idea of the problem. When you dismiss this dialog you will be shown the

expression editor with the offending expression displayed. At this point you should try to fix the problem, hit "OK", then hit "OK" on the original dialog again.

An example is in order. The module dialog, displayed when you double click on a module in an SCF window, has a priority field. If we want the priority to be taken from an exponential distribution with a mean value of 1, we can type "Exponential(1,1)" in the priority field. If we typed the name of the function incorrectly, say as "Expnential", upon clicking the "OK" button a beep would sound and an error message dialog would be displayed. After reading the message "Error: Unknown Function: Expnential" and clicking the "OK" button of the warning dialog, the Expression editor dialog would be displayed with the expression "Expnential(1,1)" ready for editing. After fixing the spelling mistake and hitting "OK" both the expression editor and the module dialogs would be dismissed. Figure 3.2 shows this succession of dialogs.



Errors in Expressions
Figure 3.2

As of this writing, the expression package will detect all errors but it is not very good at determining what the error is. Therefore, you will usually only be told that there is a syntax error. If you can detect no problems, enter "0.0" as the value so you can exit the dialog then check the manual to be sure you were passing the proper number and type of arguments to invoked functions.

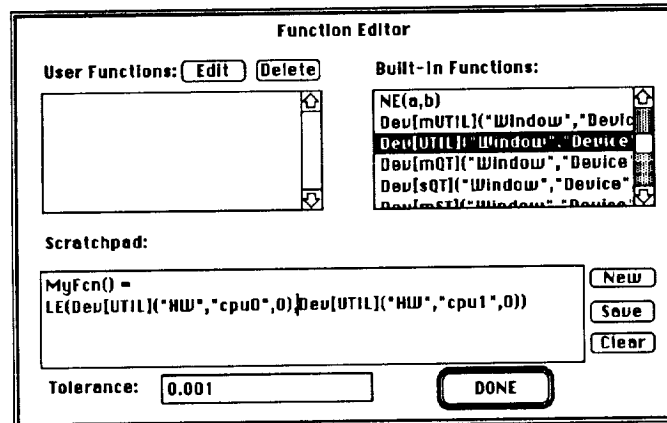
3.2. FUNCTIONS

Pedestal™ expressions may contain function calls. The functions used in expressions come from a set of predefined (Pedestal™) functions and user defined functions. The predefined functions allow, among other things, access to statistics concerning the various parts of a model and the ability to inquire/change the value of global variables.

The user defined functions allow the user to enter formulas (possibly based on other functions) in one place and then simply reference them by name in expressions. This reduces the amount of retyping necessary and may improve readability of expressions.

3.2.1. USER DEFINED FUNCTIONS

User defined functions are created in the function definition dialog. This dialog, shown in Figure 3.3, can be reached by choosing the "Functions" item of the edit menu or by the "Define New Function" button on the expression editor dialog.



The Function Definition Dialog
Figure 3.3

User functions may be defined/edited in the scratchpad area of this dialog. To enter a new function you must type the function in the scratchpad in the form:

FunctionName(ListOfParameters) = Expression

The function name can be any alphanumeric string which begins with a letter. The ListOfParameters is a list of parameter names separated by commas. These parameter names may be used in the expression which defines the new function. Note that only number parameters are allowed in user defined functions. The ListOfParameters may be empty in which case the definition will appear as:

FunctionName() = Expression

The Expression can be any valid expression. The Expression may contain invocations of built-in and/or user defined functions. To invoke an expression, either type its name or double click on it in one of the function lists in the dialog. Be sure to replace the parameters to the invoked functions with the proper values.

To have the new definition checked for errors and entered into the list of user functions if it is valid, click the "Save" button. If there is an error in the definition, an error message will be displayed which will try to explain the problem (in the current version there are only a small number of errors which are correctly explained; usually you will only be told that there is a syntax error).

If no errors are detected in the definition, the name of the function will be added to the list of user functions.

The "New" button will prompt you for a new function name and will enter the default format in the scratchpad. The "Clear" button will erase the contents of the scratchpad.

Currently defined user functions may be deleted by selecting them in the list of user functions and clicking on the "Delete" button. Since user functions may reference other user functions, when one is deleted, all references to it by other user functions are replaced by "0.0".

To edit a user function, click on the function in the list of user functions and click on the "Edit" button. This will cause the definition of the function to be displayed in the scratchpad. The definition may now be edited. To have any effect, the "Save" button must be clicked after the function is edited.

Since the "New", "Clear", "Edit" and "Done" buttons will cause the scratchpads' contents to be lost, a warning dialog will be displayed if one of these buttons is pressed when there is something typed in the scratchpad. The dialog will allow the button operation to be canceled.

The last item on the function definition dialog is the tolerance field. Since some of the built-in functions require comparisons between two real numbers, a tolerance is needed. For example, one of the parameters to the "IF" function is evaluated and compared to zero. Since in floating point representation $((10*a)/a) - a$ might not exactly equal zero, this tolerance value is used to determine how close something must be to zero for the function to consider it as zero.

3.2.2. BUILT-IN FUNCTIONS

The expression editor dialog and the function editor dialog contain built-in functions as well as user defined functions. The built-in functions are provided by Pedestal™. The definition of these functions follow. Note that the "Time Varying" notation is followed by "(?)" on some functions. Functions with this notation are not time varying unless time varying functions are used in the expressions which are given as the parameters of the function.

CLOCK()	Time Varying
Function:	
This function provides a way to determine the current time of the simulation clock.	
Parameters:	
	None.
Return Value:	
Clock() returns the current simulation time in seconds as a real.	

SETTPARM(ID,VALUE)

Time Varying(?)

Function:

During the simulation of a module, transactions (Section 4.1) move through the model. A set of 16 parameters, numbered from 0 to 15, are associated with each transaction. These parameters are provided solely for use by the user.

The SetTParm function allows the user to change the value of any of the parameters. When this function is evaluated, the transaction currently being processed will be the one whose parameter is changed.

Parameters:

	ID	<p>This expression will be evaluated and treated as an integer in the range [0,15].</p> <p>This number determines the parameter which will be modified.</p>
	Value	<p>This expression will be evaluated and treated as a real number.</p> <p>This is the value which will be placed in the IDth parameter slot of the current transaction.</p>

Return Value:

SetTParm returns the value of the IDth parameter slot of the current transaction *after* it is changed, i.e. it returns the value of the Value parameter. Value returned is a real.

GETTPARM(ID)

Time Varying(?)

Function:

During the simulation of a module, transactions (Section 4.1) move through the model. A set of 16 parameters, numbered from 0 to 15, are associated with each transaction. These parameters are provided solely for use by the user.

The GetTParm function allows the user to determine the current value of any of the parameters. When this function is evaluated, the specified parameter of the transaction currently being processed will be returned.

Parameters:

ID	This expression will be evaluated and treated as an integer in the range [0,15].
	This number determines the parameter whose value will be returned.

Return Value:

GetTParm returns the value of the IDth parameter slot of the current transaction as a real.

SETGPARM(ID,VALUE)

Time Varying(?)

Function:

A set of 256 global variables are maintained for use by the user. These variables can hold a real number.

The SetGParm function allows the user to change the value of any of the global variables.

Parameters:

ID	<p>This expression will be evaluated and treated as an integer in the range [0,255].</p> <p>This number determines the variable which will be modified.</p>
Value	<p>This expression will be evaluated and treated as a real number.</p> <p>This is the value which will be placed in the IDth global variable.</p>

Return Value:

SetGParm returns the value of the IDth global variable *after* it is changed, i.e. it returns the value of the Value parameter. Value returned is a real.

GETGPARM(ID)

Time Varying(?)

Function:

A set of 256 global variables are maintained for use by the user. These variables can hold a real number.

The GetGParm function allows the user to reference the value of any of the global variables.

Parameters:

ID

This expression will be evaluated and treated as an integer in the range [0,255].

This number determines the global variable whose value will be returned.

Return Value:

GetGParm returns the value of the IDth global variable as a real.

EXPONENTIAL(STREAM#,MEAN)		Time Varying
Function:		
Samples the exponential distribution.		
Parameters:		
stream#	This expression is evaluated as an integer in the range [0,499]. The stream number determines which random number stream is will be sampled when this function is evaluated.	
mean	The mean value of the distribution.	
Return Value:		
The exponential distributions PDF is: $F(x) = \frac{1}{\mu} e^{-\frac{x}{\mu}}$ Where: μ =MTBA (MTBA = mean value of the distribution) The Exponential function is calculated by taking the integral of the PDF and doing an inverse mapping with the number [0.,1.] from the random number stream to determine the interval. Value returned is a real.		

GEOMETRIC(STREAM#,LAMBDA)		Time Varying
Function:		
Samples the geometric distribution.		
Parameters:		
stream#	This expression is evaluated as an integer in the range [0,499]. The stream number determines which random number stream is will be sampled when this function is evaluated.	
lambda	The probability of success.	
Return Value:		
The geometric distributions PDF is: $F(n) = p(1-p)^{n-1}$ Where: p = probability of success The Geometric function is calculated by taking the integral of the PDF and doing an inverse mapping with the number [0.,1.] from the random number stream to determine the interval. Value returned is a real.		

RANDOM(STREAM#)

Time Varying

Function:

Samples one of the random number streams.

Parameters:

stream#	This expression is evaluated as an integer in the range [0,499].
	The stream number determines which random number stream is will be sampled when this function is evaluated.

Return Value:

The random number streams in Pedestal™ are guaranteed not to repeat for several million samplings.

This function will sample the designated stream and return the value as a real in the range [0.,1.].

UNIFORM(STREAM#,MIN,MAX)

Time Varying

Function:

Samples the uniform distribution.

Parameters:

stream#	<p>This expression is evaluated as an integer in the range [0,499].</p> <p>The stream number determines which random number stream is will be sampled when this function is evaluated.</p>
min	<p>This expression is evaluated as a real.</p> <p>Min designates the low end of the range for the distribution.</p>
max	<p>This expression is evaluated as a real.</p> <p>Max designates the high end of the range for the distribution.</p>

Return Value:

The uniform distributions PDF is:

$$P(x) = \frac{1}{\max - \min} \text{ for } \min \leq x \leq \max$$

$$P(x) = 0 \text{ otherwise}$$

The Uniform function is calculated by taking the integral of the PDF and doing an inverse mapping with the number [0.,1.] from the random number stream to determine the interval. The value returned is a real.

EQ(A,B)		Time Varying(?)
Function:		
This function determines if two expressions are equal. Since the expressions will be evaluated as real numbers, a tolerance must be used to determine how close the numbers must be to be considered equal. The tolerance value used is that entered in the function definition dialog (Section 3.2.1).		
Parameters:		
	a	The first expression to be evaluated and compared.
	b	The second expression to be evaluated and compared.
Return Value:		
EQ returns: 1.0 if $ a - b \leq \text{tolerance}$ 0.0 otherwise		

GE(A,B)		Time Varying(?)
Function:		
This function determines if the first of two expressions is greater than or equal to the second.		
Parameters:		
	a	The first expression to be evaluated and compared.
	b	The second expression to be evaluated and compared.
Return Value:		
GE returns:		
1.0 if a >= b		
0.0 otherwise		

GT(A,B)		Time Varying(?)
Function:		
This function determines if the first of two expressions is greater than the second.		
Parameters:		
	a	The first expression to be evaluated and compared.
	b	The second expression to be evaluated and compared.
Return Value:		
GT returns: 1.0 if a > b 0.0 otherwise		

IF(TEST,THEN,ELSE)

Time Varying(?)

Function:

This function evaluates a test expression and based on its value, evaluates one of two other expressions. The then expression will be evaluated if the test expression evaluates to a nonzero value. Since the test expression will be evaluated as real number, a tolerance must be used to determine how close to zero a number must be to be considered zero. The tolerance value entered on the define function dialog is used (Section 3.2.1).

Parameters:

test	The expression whose evaluation will determine if the then or else expression will be evaluated.
then	This expression will be evaluated if the test expression evaluates to a non-zero value.
else	This expression will be evaluated if the test expression evaluates to zero.

Return Value:

IF returns:
 {then} if |test| > tolerance
 {else} otherwise

LE(A,B)		Time Varying(?)
Function:		
This function determines if the first of two expressions is less than or equal to the second.		
Parameters:		
	a	The first expression to be evaluated and compared.
	b	The second expression to be evaluated and compared.
Return Value:		
LE returns: 1.0 if a <= b 0.0 otherwise		

LT(A,B)		Time Varying(?)
Function:		
This function determines if the first of two expressions is less than the second.		
Parameters:		
a	The first expression to be evaluated and compared.	
b	The second expression to be evaluated and compared.	
Return Value:		
LT returns: 1.0 if $a < b$ 0.0 otherwise		

NE(A,B)

Time Varying(?)

Function:

This function determines if two expressions are not equal. Since the expressions will be evaluated as real numbers, a tolerance must be used to determine how close two numbers must be to consider them equal. The tolerance value entered in the define function dialog is used (Section 3.2.1).

Parameters:

a	The first expression to be evaluated and compared.
b	The second expression to be evaluated and compared.

Return Value:

NE returns:
 1.0 if $|a - b| > \text{tolerance}$
 0.0 otherwise

DEV[MUTIL]("WINDOW","DEVICE",TComponent) Time Varying

Function:

This function determines the average utilization of any device on any hardware window (or task or lock window).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current average utilization of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[UTIL]("WINDOW","DEVICE",TComponent) Time Varying

Function:

This function determines the average utilization of any device on any hardware window (or task or lock window) during the last interval (since the last sample).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current average utilization of the device since the last sampling of this value will be returned.

See Chapter 6 for a definition of this statistic.

DEV[MQT]("WINDOW","DEVICE",TComponent)

Time Varying

Function:

This function determines the mean queue time for all jobs on any device on any hardware window (or task or lock window).

Parameters:

"Window"	<p>The name of the window which contains the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Device"	<p>The name of the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
TComponent	<p>If this value is nonzero, provide the value for the transmission component of the specified device.</p>

Return Value:

The current mean queue time of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[SQT]("WINDOW","DEVICE",TComponent)

Time Varying
Function:

This function determines the standard deviation of the queue times of any device on any hardware window (or task or lock window).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current standard deviation of the queue times of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[MST]("WINDOW","DEVICE",TComponent)

Time Varying

Function:

This function determines the mean service time of any device on any hardware window (or task or lock window).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current mean service time of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[sST]("WINDOW","DEVICE",TComponent)

Time Varying
Function:

This function determines the standard deviation of the service times of any device on any hardware window (or task or lock window).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current standard deviation of the service times of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[MRT]("WINDOW","DEVICE",TComponent)

Time Varying

Function:

This function determines the mean response time of any device on any hardware window (or task or lock window).

Parameters:

"Window"	<p>The name of the window which contains the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Device"	<p>The name of the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
TComponent	<p>If this value is nonzero, provide the value for the transmission component of the specified device.</p>

Return Value:

The current mean response time of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[SRT]("WINDOW","DEVICE",TComponent) <div style="text-align: right;">Time Varying</div>

Function:

This function determines the standard deviation of the response times of any device on any hardware window (or task or lock window).

Parameters:

"Window"	<p>The name of the window which contains the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Device"	<p>The name of the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
TComponent	<p>If this value is nonzero, provide the value for the transmission component of the specified device.</p>

Return Value:

The current standard deviation of the response times of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[#AQ]("WINDOW","DEVICE",TComponent) <div style="text-align: right;">Time Varying</div>

Function:

This function determines the total number of queue arrivals of any device on any hardware window (or task or lock window).

Parameters:

"Window"	<p>The name of the window which contains the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Device"	<p>The name of the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
TComponent	<p>If this value is nonzero, provide the value for the transmission component of the specified device.</p>

Return Value:

The current total number of queue arrivals of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[#IQ]("WINDOW","DEVICE",TComponent) Time Varying

Function:

This function determines the size of the queue of any device on any hardware window (or task or lock window).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current size of the queue of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[#DQ]("WINDOW","DEVICE",TComponent) <div style="text-align: right; font-weight: normal;">Time Varying</div>
--

Function:

This function determines the total number of departures from the queue of any device on any hardware window (or task or lock window).

Parameters:

"Window"	<p>The name of the window which contains the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Device"	<p>The name of the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
TComponent	<p>If this value is nonzero, provide the value for the transmission component of the specified device.</p>

Return Value:

The current total number of departures from the queue of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[#AS]("WINDOW","DEVICE",TComponent) <div style="text-align: right;">Time Varying</div>

Function:

This function determines the total number of work requests which have entered service of any device on any hardware window (or task or lock window).

Parameters:

"Window"	<p>The name of the window which contains the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Device"	<p>The name of the device.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
TComponent	<p>If this value is nonzero, provide the value for the transmission component of the specified device.</p>

Return Value:

The current total number of work requests which have entered service of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[#IS]("WINDOW","DEVICE",TComponent)

Time Varying

Function:

This function determines the number of work requests currently receiving service of any device on any hardware window (or task or lock window).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current work requests currently receiving service of the device will be returned.

See Chapter 6 for a definition of this statistic.

DEV[#DS]("WINDOW","DEVICE",TComponent)

Time Varying

Function:

This function determines the total number of work requests which have been serviced of any device on any hardware window (or task or lock window).

Parameters:

"Window"	The name of the window which contains the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
"Device"	The name of the device. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.
TComponent	If this value is nonzero, provide the value for the transmission component of the specified device.

Return Value:

The current total number of work requests which have been serviced by the device will be returned.

See Chapter 6 for a definition of this statistic.

MOD[MRT]("WINDOW","MODULE")		Time Varying
Function:		
This function determines the mean response time of any module/request on any SCF/RDM window.		
Parameters:		
"Window"	The name of the window which contains the module/demand. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.	
"Module"	The name of the module/demand. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.	
Return Value:		
The current mean response time of the module/demand will be returned. See Chapter 6 for a definition of this statistic.		

MOD[SRT]("WINDOW","MODULE")	Time Varying
------------------------------------	--------------

Function:

This function determines the standard deviation of the response times of any module/request on any SCF/RDM window.

Parameters:

"Window"	<p>The name of the window which contains the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Module"	<p>The name of the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>

Return Value:

The current standard deviation of the response times time of the module/demand will be returned.

See Chapter 6 for a definition of this statistic.

MOD[#AS]("WINDOW","MODULE")	Time Varying
------------------------------------	--------------

Function:

This function determines the total number of transactions which have entered any module/request on any SCF/RDM window.

Parameters:

"Window"	<p>The name of the window which contains the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Module"	<p>The name of the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>

Return Value:

The current total number of transactions which have entered the module/demand will be returned.

See Chapter 6 for a definition of this statistic.

MOD[#IS]("WINDOW","MODULE")		Time Varying
Function:		
This function determines the number of transactions in any module/request on any SCF/RDM window.		
Parameters:		
"Window"	The name of the window which contains the module/demand. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.	
"Module"	The name of the module/demand. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.	
Return Value:		
The current number of transactions in the module/demand will be returned. See Chapter 6 for a definition of this statistic.		

MOD[#DS]("WINDOW","MODULE")		Time Varying
Function:		
This function determines the number of transactions which have completed execution of any module/request on any SCF/RDM window.		
Parameters:		
"Window"	The name of the window which contains the module/demand. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.	
"Module"	The name of the module/demand. Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.	
Return Value:		
The current number of transactions which have completed execution of the module/demand will be returned. See Chapter 6 for a definition of this statistic.		

MOD[RT90%]("WINDOW","MODULE")	Time Varying
--------------------------------------	--------------

Function:

This function determines the 90th percentile of the response time of the specified module.

Parameters:

"Window"	<p>The name of the window which contains the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Module"	<p>The name of the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>

Return Value:

Returns a value such that 90% of the recorded response times for the specified module are less than the value.

MOD[RT99%]("WINDOW","MODULE")	Time Varying
--------------------------------------	--------------

Function:

This function determines the 99th percentile of the response time of the specified module.

Parameters:

"Window"	<p>The name of the window which contains the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>
"Module"	<p>The name of the module/demand.</p> <p>Case in string parameters <i>is</i> important. Additionally, the string <i>must</i> be in quotes.</p>

Return Value:

Returns a value such that 99% of the recorded response times for the specified module are less than the value.

EVENTSPERSEC()

Time Varying

Function:

This function returns a number which indicates the speed of the simulator. It is of use for comparing the speed of different machines and using/not using multifinder. This value can be used to determine which platform is best for running large simulations.

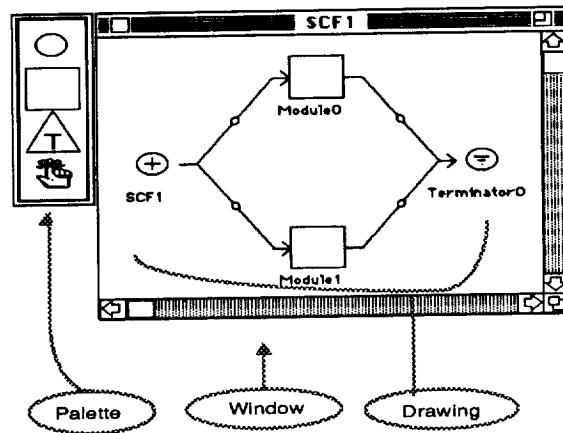
Parameters: none.**Return Value:**

The current average number of events processed per second is returned as a real.

PAUSE("MESSAGE")		Time Varying
Function:		
This function causes a dialog box to be displayed which will contain the supplied "Message" and then puts the simulator in single step mode (Section 2.7).		
Parameters:		
"Message"	The string to be displayed in the message dialog box when this pause statement is executed.	
Return Value:		
0.0		

4. PEDESTAL MODELS

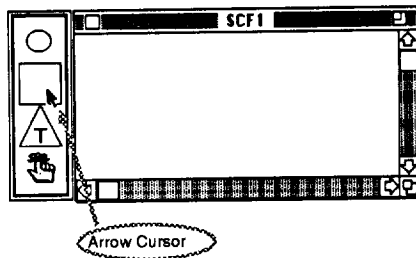
Model specification in Pedestal™ is done primarily in the form of drawings. These drawings define the individual hardware and software components of a model. Navigation among the various window can be accomplished through the "Desktop" window which allows access to all the diagrams which comprise a model. Relationships between software and hardware components are determined via the mappings described in Chapter 5. **This approach allows the software and hardware components of a model to be specified independently.**



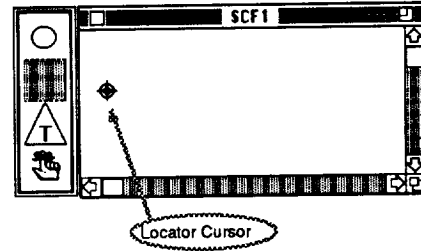
A Pedestal™ Window
Figure 4.1

Most windows look similar to the one shown in Figure 4.1. The palette to the left of the window contains icons which represent the basic descriptive units allowed on the specific window. Drawings are created by placing icons on the window and drawing connectors between them. To place an icon from the palette onto the window, click on the desired palette icon. The icon in the palette will then be highlighted. Notice that when a palette icon is selected, the cursor assumes the "locator" form when within the windows contents. Whenever the cursor is in the "locator" form, clicking on the window will place something in the window. Click the mouse with the cursor positioned where you want the icon to appear and the icon will be placed. Icons within a window may be referred to as object icons. After placing the icon in the window, the cursor will return to the arrow form. Figures 4.2 - 4.4 show the process of picking an icon from the palette and placing it in the window.

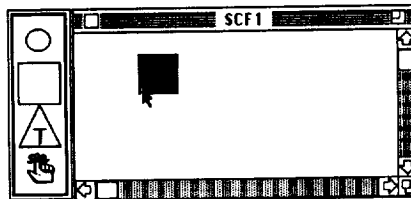
If you wish to place another icon of the same type as you just placed, hold the command key (⌘) and click the mouse in the window. The mouse will be changed to the locator form and you can then click to place the icon. Also, if the cursor is in the locator form and you do not want to place, hold the command key (⌘) and click the mouse in the window. The cursor will be changed back to the arrow and nothing will be placed.



Before Clicking on the Palette
Figure 4.2



After Clicking on the Palette
Figure 4.3



After Placing the Icon
Figure 4.4

To draw a connector between two icons, hold the command key (⌘) down, position the cursor over one of the icons and press the mouse button. Now, with the command key (⌘) still depressed, drag the cursor until it is over the other icon. A dashed line will be drawn from the first icon to the cursor as it is moved. When the cursor is over the other icon, release the mouse button and the icons will be connected. Pedestal™ determines how the connectors will be drawn depending on the relative positioning of the icons and independent of the path over which the mouse is dragged. When either of the icons is moved, the connector will be redrawn.

Within a window, icons and connectors may be selected individually or in groups. Individual selection is the same as on the palette, i.e. position the cursor over the icon and click the mouse. Multiple selection is accomplished by positioning the cursor in one corner of a rectangular area that is to be selected. As the cursor is dragged a broken line will be drawn which delineates the rectangular area in which all icons and connectors will be selected. Operations such as cut, copy, paste and delete may be performed on the entire group of selected items by selecting the proper commands from the Edit Menu or using the keyboard commands associated with these operations. For further explanation of these standard Macintosh functions refer to the documentation provided with your computer.

Icons, connectors and windows are basic building blocks in the Pedestal™ user interface. Actions which may be performed on the differing icons is common to all windows, with the results being context sensitive. For example, once placed on a window any icon can be double clicked; the effect of double clicking may be opening of another window, display of a dialog box or some other result that depends on the icon type. This commonality of actions

with possibly different effects led us to implement the window system in an object oriented way to reduce the complexity of the code while enhancing its robustness.

The implementation of the interface is done using a set of fields and method pointers for each window which define how that specific window acts and is acted upon. Since only one type of connector is allowed on a particular window, some of the fields and methods on the window also deal with the connectors for that window. The fields for a window are shown below with a brief description of the significance of each:

WindowType	Determines the palette of icons to display when the window is active.
ConnectorType	Determines how to connectors are drawn in the window - may be directed or not.
CanEdit	Determines if editing (cut/paste/etc) is allowed in the window.
CanGetInfo	Determines if the "Get Info" item of the Edit menu (see Section 2.3) is allowed in the window.
CanMouse	Determines if the user is allowed to move icons and create connectors in the window.
CanReduce	Determines if the "Reduce To Fit" item of the Draw menu (see Section 2.4) is allowed in the window.
CanTile	Determines if the tiling items of the Draw menu (see Section 2.4) affect the window.

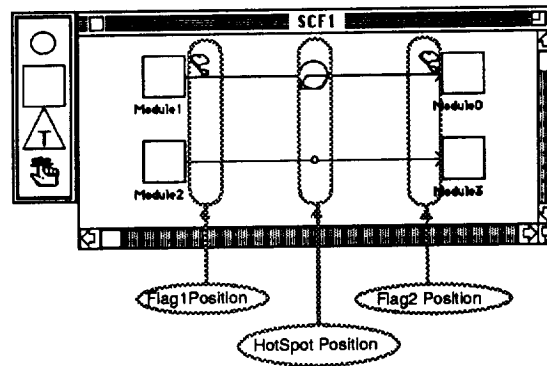
The specific settings of these fields will be explained later in this Chapter as each type of window is discussed individually. The method pointers determine what happens on the window in response to an action and are briefly outlined below:

ConnectorDoubleClick	Called when the user double clicks on a connector.
PlaceConnector	Called when the user creates a new connector in the window.
Activate	Called when a window is activated.

More specific information regarding the methods will be provided in the discussion of individual window types.

Fields and methods are not unique to windows but are also applied for icons and connectors as they are created. The fields for a connector are:

Flag1Icon	Determines the icon (if there is one) to display in the first flag position (see Figure 4.5).
Flag2Icon	Determines the icon, if any, to display in the second flag position (see Figure 4.5).
HotSpotIcon	Determines the icon, if any, to display in the hotspot position (see Figure 4.5).



Connector Variations
Figure 4.5

The specific settings of these fields will be explained in the rest of this Chapter for each specific type of window. The fields for an icon are:

Genus	Determines the family of the icon.
Species	Determines the specific icon of the given family (genus) to be displayed for this icon - a visual aid to specialization.
Type	Determines if the icon is a block icon or a line icon - A block icon appears on the window the same as it does in the palette, i.e. it is of a fixed size. A line icon can have its shape and size altered on the window.
Shadow	Determines if the icon is to be drawn with a drop shadow.

The connector and icon methods are very similar and are treated together here with the few differences noted below. The method pointers for icons and connectors are:

Deleted	Called when an object is deleted - the item is deleted when it is selected and the user chooses "Cut" from the Edit menu (Section 2.3) or hits the delete key on the keyboard. Note that Pedestal™ can initiate a deletion on its own, e.g. when the user opens a new model, the old model is deleted automatically.
Duplicated	Called when an object is duplicated by the user selecting the "Cut" or "Copy" command from the Edit menu (Section 2.3) - a copy of the selection is placed in the Clipboard. As in Deletion, Pedestal™ may initiate a duplication while performing another operation.
Pasted	Called when the user selects the "Paste" command on the Edit menu (Section 2.3) and there is something in the Clipboard. The contents of the Clipboard are copied into the current window if its contents are compatible. Again, Pedestal™ may initiate a paste operation while performing another operation.
Gender	Called for a connector after a model has been read from a file. For an icon, this method is called whenever its

	species is changed, after a model has been read and when a connector is attached or detached from the icon.
Save	Called for all objects when a model is saved by the user choosing the "Save..." command of the file menu (Section 2.2)
Restore	Called for all connectors and/or icons when a model is restored (read). To restore, the user must choose the "Open..." command of the file menu (Section 2.2).
Connected	This method is called only for icons . It is called when an attempt is being made to draw a connector to or from the icon.
DoubleClick	Called for objects when the user double clicks on an icon or connector in a window.
GetInfo	This method is called only for icons . It is called when an icon is selected and the user chooses the "Get Info" item from the Edit menu (see Section 2.3). Note: this is only true if the windows CanGetInfo flag is set to TRUE.

4.1. SOFTWARE

A Pedestal™ model consists of both a description of a hardware system and a set of software processes which are intended to run on the hardware. The software model consists of individual resource demands (execute x instructions, read x bytes, etc.) and the flow of control between these resource demands. A group of these demands and the control flows between them are grouped into Resource Demand Modules (RDMs). RDMs by definition execute as a unit, i.e. all execution demands within an RDM must execute on the same device. This does not mean that different instances of an RDM are restricted to a single device but only that each instance is indivisible for execution purposes. When the software is partitioned into groups of code, each of which will execute on a particular device, the RDM is the smallest unit which can be partitioned. RDMs are defined in Pedestal™ either by textual specification or by drawings. The drawing method uses a separate window for each RDM, with icons representing the resource demands and directed connectors used to denote the flow of control. RDMs in textual form may have no internal parallelism since they are defined as a sequential list of resource demands. RDM windows and textual RDMs are described in more detail below.

Moving up a level in the Pedestal™ software model we have the Software Control Flow (SCF). An SCF is comprised of a stimulus, a set of software modules and the data flows between them. A stimulus is a user defined group of workloads which initiate data flows in an SCF. A software module is an instance of an RDM; multiple modules may reference the same RDM. The module determines the priority at which the RDM instance will be executed and specifies the task to be used. A task is a group of modules which will execute on the same device, sometimes referred to as a partition. Each SCF is defined in a separate window, with its own drawing. Connectors are used to denote the flow of data (and control) within the SCF. A special icon in the SCF palette allows SCF windows to be nested. These nested SCFs are referred to as macros. SCF and Macro windows are described in more detail below.

4.1.1. SOFTWARE CONTROL FLOWS

The user interface of windows, icons and connectors that is described in the beginning of this Chapter provides the tools with which a Pedestal™ user builds a model. A model may be thought of as a multifaceted problem that is to be analyzed. A HOW, WHERE, WHAT and WHEN approach to the problem may help to explain the inter-relational aspects of Pedestal™ model. The Software Control Flow may be perceived as the "WHAT" of the problem in contrast to the "WHERE" which is specified in the hardware diagram. "HOW" is determined by the mappings which define task composition, processor usage and file residence. "WHEN" is determined by the workloads in the SCFs.

During simulation workloads in the stimulus periodically create new transactions. A transaction is an input to the system which will pass through the SCF being processed by modules, delayed by delays and passed along data flows until it reaches a terminator. A process can have several outgoing dataflows. After processing a transaction, a process will send a transaction along one of its outgoing data flows. The process may also attempt to send copies of the transaction along one or more of the remaining outgoing data flows. Whether the process produces one or many outgoing transactions per incoming transaction is determined by the setting of the "Output Fission" attribute of the process. If the fission box is checked on an icon's dialog box then the icon will always send the transaction along one of the data flows and will also attempt to send a copy of the transaction along each of the remaining outgoing data flows. If the fission box is not checked, only the original transaction will be sent, and it will be sent along only one of the data flows.

Each data flow has a set of branching probabilities associated with it. This set contains a probability for each workload defined in the SCF stimulus. Transactions carry information about their associated workload and whether they are a copy or an original transaction. In the absence of fission, processing of a transaction is followed by a decision as to which path the transaction will follow. Each outgoing data flow's branching probability for the workload which created the transaction will be compared to a random number between zero and one. The first data flow whose branching probability for that workload is greater than the random number will be used to transmit the transaction. The probability from a failed comparison will be added to the random number used in the next comparison. Note that it is assumed that the sum of the probabilities pertaining to a given workload on all data flows leaving an icon is 1.

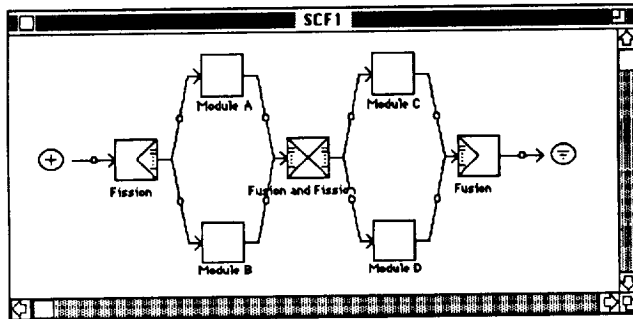
If fission is being used, a transaction is processed and then each outgoing data flow's branching probability for the workload which created the transaction will be compared to a random number. A new random number will be used for each comparison. Each data flow whose branching probability for that workload is greater than the random number will have a transaction sent along it. The last data flow whose comparison is successful will receive the original transaction, all other (successful) data flows will receive copies.

Whether fission is used or not, the original transaction **will** be sent along one data flow. If all comparisons fail, the last data flow checked will be used to transmit the transaction. The outgoing data flows are tested in priority order (priority is specified on the connectors dialog). Since the probabilities in the branching list are entered as expressions (Chapter 3) it is possible to have rule based routing as well as simple probabilistic routing.

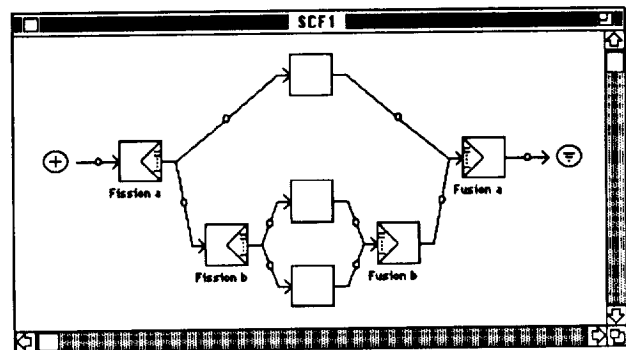
A data flow concept linked to fission is fusion. A process downstream from where a fission occurs may queue arriving transactions until all copies of that transaction which can reach it are queued. Then the copies will be deleted and the process will be activated to process the original (single) transaction. The terminator process on an SCF window always performs fusion. Figure 4.6 and 4.7 are examples of fusion and fission use. The names of the modules and the appearance of the icons on the SCF denote where fission and/or fusion are performed. In Figure 4.6, two sequential fusion/fission pairs are shown. Assuming that only one transaction can be processed at a time and all branching probabilities are 1, the processing of a transaction would proceed as follows:

1	A transaction T1 is generated by a workload in the stimulus and is sent to Module Fission.
2	Module Fission processes T1.
3	Module Fission performs a fission. A copy of T1 called T1.1 is made. T1 is sent to Module A. T1.1 is sent to Module B.
4	Module A processes T1 and sends it to Module "Fusion and Fission".
5	Module "Fusion and Fission" performs a fission: T1.1 is queued.
6	Module B processes T1.1 and sends it to module "Fusion and Fission".
7	Module "Fusion and Fission" performs a fission: T1 is queued. T1.1 is deleted. T1 is dequeued.
8	Module "Fusion and Fission" processes T1.
9	The remainder of the processing is the same as above (from step 3 on).

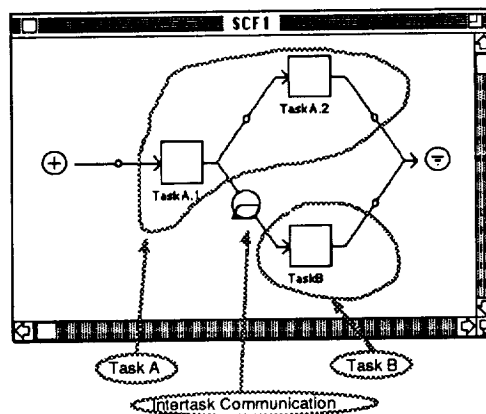
Modules are assigned to tasks which in turn are mapped to execution devices. If a data flow occurs between modules in different tasks, a message must be sent between the corresponding processing devices (assuming that the tasks are assigned to different processors). The connector is used to determine the priority and size of such a message. Thus, in a sense, the size of a transaction can vary. To indicate to the user that interprocessor communication may occur along a given data flow, the corresponding connector will be drawn in a slightly different fashion. Figure 4.8 shows an example of such a connector (between modules TaskA.1 and TaskB).



Fusion/Fission Example
Figure 4.6



Nested Fusion/Fission Pairs
Figure 4.7

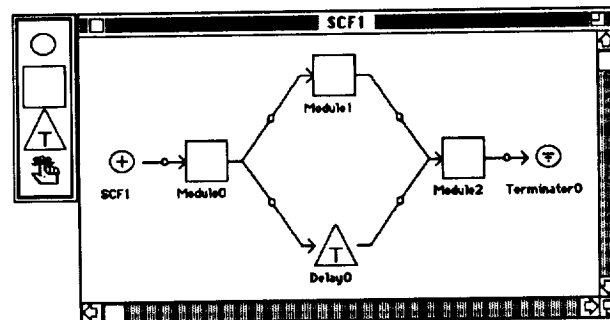


Intertask Communication Notification on Connectors
Figure 4.8

4.1.1.1. SCF & MACRO WINDOWS

SCF windows provide a top level description of the software being modeled. Macro windows are the same as SCF windows except that they are nested within an SCF or Macro window and are associated with an icon on the parent window. This nesting of windows allows the software architecture to be defined in a top down hierarchical manner. Existing icons may not be grouped into a macro; rather, the module must be declared to be a macro and the contents of the macro subsequently defined. The icons in the SCF palette represent a stimulus (or terminator), module (or macro) and a delay (for an event which is not modeled). The last icon is the note icon; this icon allows the user to attach any notes or documentation to the diagram. More specific descriptions of the icons in the SCF windows will be given in following Sections. Figure 4.9 shows a typical SCF window.

SCF windows are created by choosing the "New SCF" item from the draw menu (Section 2.4). All currently defined SCF windows appear in the software folder window and may be accessed by double clicking the corresponding icon on the desktop window. All open windows are also listed at the bottom of the draw menu and may be brought to the front by selecting them from that menu.



An SCF Window
Figure 4.9

The object fields for the SCF window are set as follows:

Field Name	Setting
WindowType	SCF {or Macro}
ConnectorType	HorizontalArrow {Varies}
CanEdit	True
CanGetInfo	True
CanMouse	True
CanReduce	True
CanTile	True

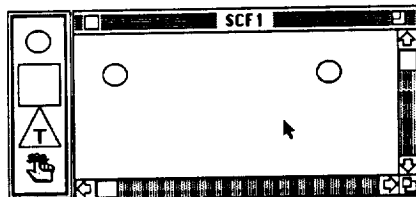
Connectors are also supported on the SCF window, thus some of the window methods are used:

Method Name	Function
ConnectorDoubleClick	Display the data flow dialog (see below).
PlaceConnector	Creates a new data flow.
Activate	No function.

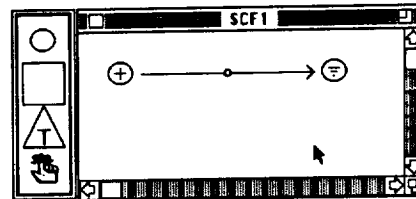
4.1.1.2.

THE STIMULUS/TERMINATOR ICON

As previously mentioned, there must be one and only one each stimulus and terminator on an SCF window. The stimulus holds the workload descriptions and the terminator is used as the final destination of all transactions. To save space on the palette, one icon appears there to represent both of these icons. When the user connects to or from the generic icon, its species is changed and it becomes a terminator or stimulus depending upon the direction of the connection. This context sensitive gendering gives visual reinforcement to the diagram content. Figure 4.10 shows two stimulus/terminator icons placed in a window. Figure 4.11 shows the species of the stimulus and terminator after the two generic icons are connected (the stimulus is on the left). Note that connectors may only be drawn to terminators, not from them. Similarly, connectors may only be drawn from a stimulus, not to it.



Stimulus/Terminator Icons Before Connecting
Figure 4.10



Stimulus/Terminator Icons After Connecting
Figure 4.11

This icon has its object fields set to:

Field Name	Setting
Genus	Stim/Term
Species	Stim or Term
Type	BlockIcon
Shadow	False

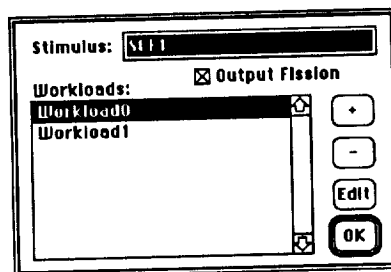
As mentioned above, the species of this icon changes to stimulus or terminator as soon as a connection is made to it. Shadowing is not used in SCF windows.

The methods for the stimulus/terminator icon are:

Method Name	Function
Deleted	Destroys the data structure associated with the icon. The stimulus data structure is a list of workloads. The terminator has no data structure.
Duplicated	Not allowed.

Pasted	Not allowed.
Gender	When a connection is made to a stimulus/terminator icon, the gender routine is activated and the species is set according to the direction of the connector. This routine also ensures that once the icon has its species, e.g. stimulus, that connectors are only made to it in the correct direction. I.E. stimulus icons may only be connected from not to.
Save	Saves the data structure associated with the icon to the model file.
Restore	Reads a data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	Only allows connectors to be drawn from stimulus icons and only allows connectors to be drawn to terminator icons.
DoubleClick	Double clicking a stimulus will cause the workload list dialog to be displayed (see below). Double clicking on a terminator will cause a simple dialog to be displayed allowing the name of the terminator to be changed.
GetInfo	Same as double click (see above).

As mentioned in the method list, double clicking on a stimulus will display the workload list dialog shown in Figure 4.12. The stimulus name field allows the user to change the name of the stimulus icon and thereby the name of the SCF window. This name must be distinct from all other window names in the model. The "Output Fission" checkbox determines if fission is performed at the stimulus. The default is that fission is performed.



The Workload List Dialog
Figure 4.12

The list of workloads in the workload list dialog displays the name(s) of all currently defined workloads in this stimulus (read SCF). By default, there will be one workload named "Workload0" defined. To create a new workload click

on the "+" button. This will cause a dialog to be displayed with a default name that can be edited if desired. Once the new name is entered, you are returned to the workload list dialog and the new workload will be added to the list. To delete a workload, select its name in the list by clicking on it once and then click on the "-" button. To view/edit the definition of a workload, select the name of the workload in the list by clicking on it once with the mouse, then click on the "Edit" button. Alternatively, you may edit a workload by double clicking on the workload name in the list. The "OK" button dismisses the workload list dialog.

When you edit a workload either by double clicking on its name in the workload list or selecting it and clicking on the "Edit" button, the workload dialog will be displayed as shown in Figure 4.13. The workload name may be changed by editing the current name in the workload name field. Clicking on the "Edit Start/Stop Conditions" button will display the start/stop time dialog described below. A workload defines the entry of stimuli (work requests) into an SCF. The workload interarrival time determines the time delay between stimuli arrivals. The interarrival time is specified as a distribution. The distribution popup lists the valid distributions. After selecting the desired distribution (default is exponential), the parameters to the distribution will be displayed in the parameter field. To edit these parameters, click on the parameter field and a dialog will be displayed which lists the parameters and allows their values to be edited. Figure 4.14 is an example of this dialog which is displayed for the exponential distribution. The number and names of the parameters will vary depending on the distribution currently showing in the distribution popup. The time unit popup, determines the time scale.

The Workload Dialog (Figure 4.13) is a window with the following elements:

- Workload:** A text field containing "Workload0".
- Edit Start/Stop Conditions:** A button.
- Priority:** A text field containing "1".
- Interarrival Time:** A section containing:
 - Exponential:** A button.
 - (1,1.0):** A text field.
 - seconds:** A text field.
- Batch Size:** A section containing:
 - Constant:** A button.
 - (1):** A text field.
- Buttons:** "Cancel" and "OK" buttons at the bottom right.

The Workload Dialog
Figure 4.13

A Distribution Parameter Dialog (Figure 4.14) is a window with the following elements:

- Arrival Distribution:** A section containing:
 - Exponential:** A button.
- Parameter Name** and **Value** headers.
- Parameters:**
 - 1. Stream#: A text field.
 - 2. MTBA: A text field containing "1.0".
- Buttons:** "Cancel" and "OK" buttons at the bottom right.

A Distribution Parameter Dialog
Figure 4.14

Each time a stimulus arrives, more than one transaction may be created. The batch size of the workload determines how many transactions are created when an arrival occurs. This value is specified, just as the interarrival time distribution, from a popup menu. The only difference is that there is no need of a unit scale for batch size.

As mentioned above, clicking on the "Edit Start/Stop Conditions" button on the workload dialog displays the start/stop time dialog. Figure 4.15, shows this dialog. The start time determines when this workload becomes active and starts generating arrivals. The stop condition must be chosen by clicking on one of the three radio buttons. The "End of Simulation" stop condition is self explanatory. The "After x stimuli generated" condition, if chosen, will turn off the

workload once the stated number of transactions have been generated by this specific workload. The "At Time x" condition, if chosen, will turn off the workload once the simulation reaches the specified time.

The Start/Stop Time Dialog
Figure 4.15

The values shown in fields concerning the workload and start/stop time dialogs are the default values of these fields.

Both the interarrival time and the batch size are defined as distributions. The distributions which appear in the popup menus, their parameters and their formulas are:

Distribution	Parameters {defaults}	Probability Density Function
Random	Stream {1}	Next random number in stream.
Uniform	Stream {1} Min {0} Max {1}	$F(x) = \frac{1}{\max - \min} \text{ for } \min \leq x \leq \max$ $F(x) = 0 \text{ otherwise}$
Exponential	Stream {1} Mean Time Between Arrivals {1}	$F(x) = \frac{1}{\mu} e^{-\frac{x}{\mu}} \text{ Where: } \mu = \text{MTBA}$
Geometric	Stream {1} Probability of Success {0.5}	$F(n) = p(1-p)^{n-1}$ <p>Where: p = probability of success</p>
Constant	Value {1}	$F(x) = x$

The actual values are calculated by taking the integral of the PDF and doing an inverse mapping with a generated random number on [0,1.0] to determine the x value.

$r = \text{random number on } [0,1.0]$

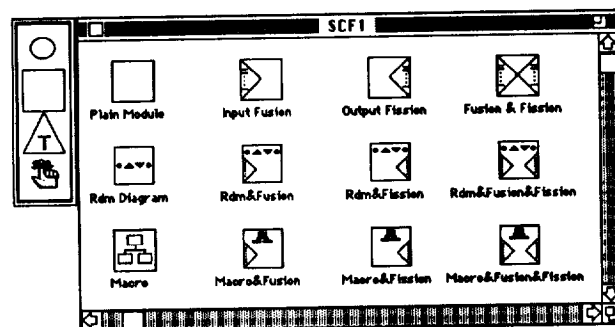
given $r = \int_{-\infty}^x f(x)dx$

What is x ?

There are 500 (0 - 499) random streams available to the user of Pedestal™, each of the which is guaranteed not to repeat in less than one million samples.

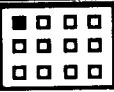
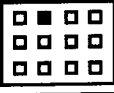
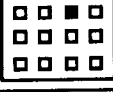
4.1.1.3. THE MODULE ICON

The main components of an SCF window are software modules and the control/data flows between them. The module icon in the SCF palette is used to represent a software module. There are two basic types of software modules; simple modules and macros. Simple modules represent a group of resource demand statements referred as a Resource Demand Module (RDM) which must be partitioned as a group. The RDM which is associated with a module may be defined either as a textual list of resource demands where order in the list denotes control flow, or it may be defined as an RDM window. RDMs are explained in more detail in Section 4.1.2. The macro version of a module represents a nested SCF diagram composed of one or more modules. The macro is provided to allow hierarchical representation of the software architecture of the model, thereby reducing the complexity of the SCF windows. Modules can also affect control flow by performing input fusion and/or output fission. The alternate iconic forms (species) denoting functional or structural particulars of the module are shown in Figure 4.16.



The Module Species
Figure 4.16

The different species denote whether the module has a textual RDM, an RDM diagram, or is a macro. Additionally, each of these base types has species which denote the presence of fusion and/or fission. In the following table the leftmost column represents Figure 4.16 with the position of the icon being described highlighted. The meaning of the species are:

Icon	Macro	RDM	Fusion	Fission
	No	Text	No	No
	No	Text	Yes	No
	No	Text	No	Yes

	No	Text	Yes	Yes
	No	Diagram	No	No
	No	Diagram	Yes	No
	No	Diagram	No	Yes
	No	Diagram	Yes	Yes
	Yes	No	No	No
	Yes	No	Yes	No
	Yes	No	No	Yes
	Yes	No	Yes	Yes

The species of a module icon is changed automatically by Pedestal™ to match the current definition of the module. If the user changes the definition, Pedestal™ will automatically regenerate the icon and thereby change its species to match the new definition.

One of the main factors in the performance of a multiprocessor system is partitioning of the software to the hardware, i.e. what software executes on what processor. In Pedestal™, simple modules (not the macro type) are the smallest partitionable unit. Tasks are used to denote partitioning. Modules are assigned to tasks and tasks are later assigned to processors. Tasks are discussed in more detail in Section 4.2. Mapping is the term used for the assignments mentioned above and is discussed in more detail in Chapter 5.

The module icon has its object fields set to:

Field Name	Setting
Genus	Module
Species	Module {Variable}
Type	BlockIcon
Shadow	False

As mentioned above, the species of this icon varies between several values depending on the characteristics of the specific module. Shadowing is not used in SCF windows.

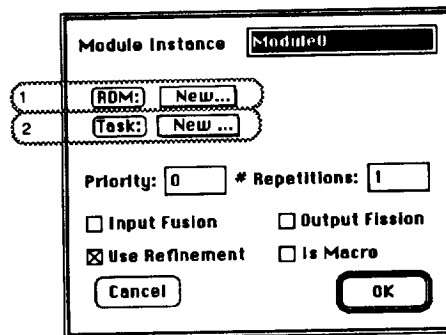
The methods for the module icon are:

Method Name	Function
Deleted	Destroys the module data structure associated with the icon.
Duplicated	Attaches a module data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated. However, the name will be changed so that it is unique from all other names in the Clipboard.
Pasted	Ensures that the name of the module data structure associated with the icon being pasted is distinct from all other modules in the window being pasted into. The name will be changed if necessary.
Gender	After the user edits the module dialog the icon is regenerated so that its species properly reflects the new characteristics of the module.
Save	Saves the module data structure associated with the icon to the model file.
Restore	Reads a module data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking a module icon will cause the module dialog (see below) to be displayed.
GetInfo	A module will have either a textual RDM, RDM diagram, or a Macro (SCF) window associated with it. GetInfo will cause either the textual RDM dialog (Section 4.1.2.2) or the RDM or Macro window to be displayed as appropriate.

As mentioned in the method list, double clicking on a module icon will display the module dialog. Figure 4.17 shows a sample module dialog. The "Module Instance" field allows the module name to be edited. The names of all icons on a SCF window must be unique.

Since the same software routine may occur in multiple places on an SCF, a module may reference any RDM, even one that other modules reference. The body of the RDM only needs to be entered once and any changes to the RDM will be carried to all referent modules. Region 1 in the

module dialog allows a module's RDM to be specified and/or edited. The popup menu will contain the names of all currently defined RDMs and will display the name of the RDM currently associated with the module. Additionally, the popup menu will have a "New..." entry, which appears prior to the selection of an RDM. Upon choosing this first entry, a new RDM is created with a unique name. The new RDM will then appear as the current choice in the popup. For a given RDM name there may exist both a diagram and a textual specification. The text version is meant to be used as a simpler version of the diagram with the same name. Whether the current module is associated with the text or the diagram version of the RDM named in the popup is determined by the "Use Refinement" check box on the module dialog. If this box is checked, the diagram is used, otherwise the textual version is used. The purpose of the "RDM:" button in region 1 of the dialog is to display the RDM associated with the module so that it may be viewed and/or edited. The setting of the "Use Refinement" check box will determine whether this button displays the RDM text dialog or the specified RDM window. See Section 4.1.2 for more information regarding RDMs.



The Module Dialog
Figure 4.17

Region 2 of the module dialog indicates the task to which the module is assigned. The popup menu will display the name of the task currently associated with the module. The popup will contain names of all currently defined tasks. Additionally, the popup will have a "New..." item. Choosing this item will cause a new task with a unique name to be created and associated with this module. Clicking on the "Task:" button will display the task dialog for the task associated with the module so that it may be edited/viewed. For more information regarding tasks see Section 4.2.

The "Priority" field of the module dialog sets the priority of this module, one criteria by which the order of processing of transactions may be determined. The operating system (manager) of the device to which a task is mapped determines if priority is used and if so, which type (module, task or workload). For more information about operating systems see Section 4.6. For more information about software to hardware mapping see Chapter 5.

"# Repetitions" determines the number of times the body (RDM) of the module is executed for each transaction which arrives at this module. The

transaction will be routed through the RDM the specified number of times before exiting the module.

The remaining fields on the module dialog are all check boxes. The function of the "Use Refinement" check box has already been discussed for non macro modules. For macro modules, the box determines whether the macro is translated as a macro or as the text version of the RDM named in the RDM popup (in region 1) when the model is simulated. If the box is checked, the module will be translated as a macro. The "Is Macro" check box is used to turn the module into a macro. If checked, the module will be treated as a macro. The "Input Fusion" and "Output Fission" check boxes, if checked, denote the presence of the respective operation on this module.

The module dialog shown in Figure 4.17 has the default settings for a new module icon.

4.1.1.4. THE DELAY ICON

The delay icon is used to represent delays caused in the system by events not being explicitly simulated in Pedestal™. The delay icon has its object fields set to:

Field Name	Setting
Genus	Delay
Species	Delay
Type	BlockIcon
Shadow	False

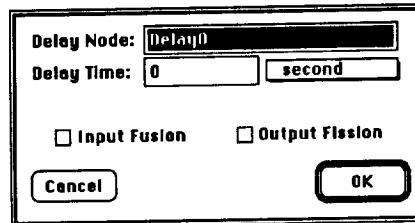
There is only one species of the Delay icon. Shadowing is not used in SCF windows.

The methods for the delay icon are:

Method Name	Function
Deleted	Destroys the delay data structure associated with the icon.
Duplicated	Attaches a delay data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated. However, the name will be changed so that it is unique from all other names in the Clipboard.
Pasted	Assures that the name of the delay data structure associated with the icon being pasted is distinct from all other modules in the window being pasted into. The name will be changed if necessary.
Gender	No Function.
Save	Saves the delay data structure associated with the icon to the model file.
Restore	Reads a delay data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking a delay icon will cause the delay dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on a delay icon will display the delay dialog. Figure 4.18 shows a sample delay dialog. The name of the delay

may be edited in the "Delay Node" field. The name of the delay must be distinct from the name of all other icons in the SCF window. The length of the delay is determined by the "Delay Time" field. The time unit popup provides the scale of the delay. It should be noted that the use of expressions allows the length of the delay to change over time. The "Input Fusion" and "Output Fission" check boxes denote the absence (if unchecked) or presence (if checked) of the associated operations on this icon.



The Delay Dialog
Figure 4.18

The delay dialog in Figure 4.18 shows the default settings for a new delay.

4.1.1.5.



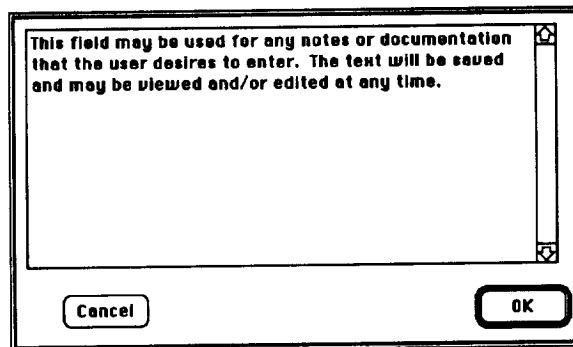
THE NOTE ICON

The note icon appears on all windows in Pedestal™ except for the desktop and hardware and software folder windows. This icon allows the user to attach notes and/or documentation to Pedestal™ diagrams. This icon has its object fields set to:

Field Name	Setting
Genus	Note
Species	Note
Type	BlockIcon
Shadow	False

The methods for the note icon are:

Method Name	Function
Deleted	Delete the text associated with the icon.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	No function.
Save	Saves the text associated with the icon to the model file.
Restore	Reads a text from the model file and associates it with the proper window and icon.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the note dialog to be displayed (see below).
GetInfo	No function.



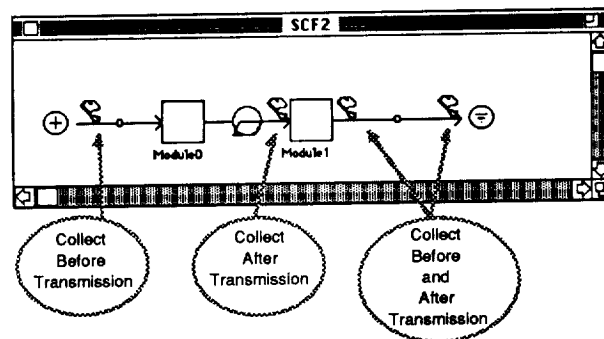
The Note Dialog
Figure 4.19

As mentioned in the method list, double clicking on a note icon will display the note dialog shown in Figure 4.19. This dialog contains a scrollable text field. The user may type in this field and the information will be saved for subsequent viewing and editing.

4.1.1.6. SCF/MACRO CONNECTORS

The connectors on an SCF window are used to denote existence and direction of data flow paths. These connectors determine the flow of work throughout the SCF. The note icon has nothing to do with the simulation of software and therefore can not have data flows to or from it. The stimulus icon can only have connectors drawn from it and the terminator can only have incoming connectors. The module and delay icons can have connectors both to and from them. In order to be part of a running model, a module or delay must have both an incoming and outgoing connector.

The HotSpotIcon area of SCF connectors is used to denote inter-task boundaries. Another use of the connectors on SCF windows is in statistics collection. The user is allowed to define up to 4 statistics to be collected for each data flow. These statistics will be collected each time a transaction is about to be sent along the given data flow. A small flag icon will be attached to the start and/or end of a connector to denote the presence of statistics collection. Figure 4.20 shows an example of these modified connectors.



Statistics Collection Notification on Connectors
Figure 4.20

SCF connectors always appear as directed arrows and as mentioned above, have various appearances. The object field settings for connectors are:

Field Name	Setting
Flag1Icon	{None,Flag}
Flag2Icon	{None,Flag}
HotSpotIcon	{None,ITC}

The flags are used to denote presence of statistics collection on connectors and the ITC symbol is used to denote the presence of intertask communications. The method routines for a connector are:

Method Name	Function
Deleted	Destroys the branching list and statistics information associated with the connector.

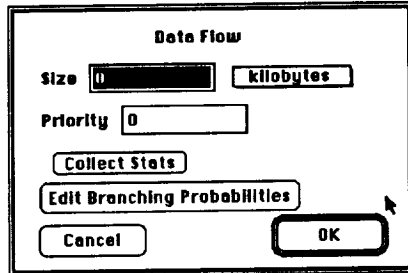
Duplicated	Attaches a branching list and set of statistics to the new duplicate connector. The statistics will be the same as the original connector. The branching list will be recreated (with possibly different probabilities) since the connector may be pasted to an SCF window with different workloads.
Pasted	If the icons on both ends of the connector are also pasted then the connector is pasted and the branching probabilities are changed so that the workloads match this window.
Gender	Determines if statistics are defined for this connector before or after transmission and if the icons on each end of it are in different tasks. Based on this, the connector is drawn with the proper modifications.
Save	Saves the branching list and statistics information associated with the connector to the model file.
Restore	Reads a branching list and statistics information from the model file and recreates the structure. This structure will then be associated with the proper connector.
DoubleClick	Double clicking a connector will display the data flow dialog (see below).

As mentioned in the method list, double clicking on a connector will display the data flow dialog. This dialog, shown in Figure 4.21 allows the information associated with the connector to be viewed/edited. The size field and popup menu determine the amount of data to be sent as a transaction passes along the connector. The priority field determines the transmission priority of the transaction. The "Collect Stats" button displays the connector statistics dialog (explained below) for the statistics to be collected before transmission of a transaction along this data flow. Finally the "Edit Branching Probabilities" button will display the branching probability dialog (explained below) to allow the probabilities for this connector to be edited.

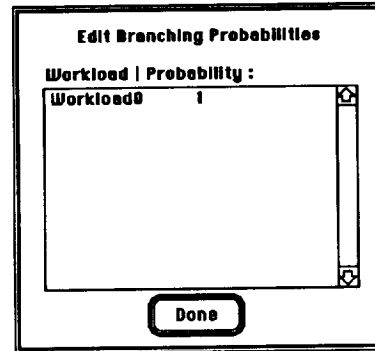
The branching probability dialog, shown in Figure 4.22, contains a scrolling list of workload name/probability pairs. An entry will appear in this list for every workload defined in the stimulus of the SCF window in which the connector is drawn. To change a probability, double click on the number and a simple dialog will be displayed which will prompt you for the new probability and update the list.

The connector statistics dialog, shown in Figure 4.23, allows four statistics to be defined. The label field allows the user to specify the name of the statistic to be calculated. The expression field allows the user to enter an expression which defines the statistic. The show box, if clicked, will enable the

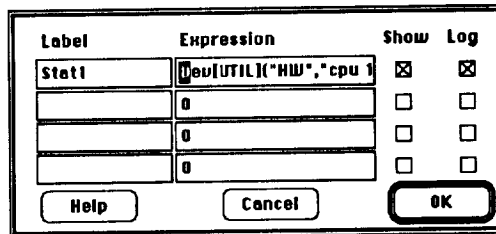
statistic to be shown on the screen during a simulation run if the Show SCF Stats is selected on the. Finally, the log box, if clicked will cause the statistic to be written out to the event log file each time it is evaluated during a simulation run. The event log file is specified on the collect file statistics dialog explained in Chapter 6.



The Data Flow Dialog
Figure 4.21



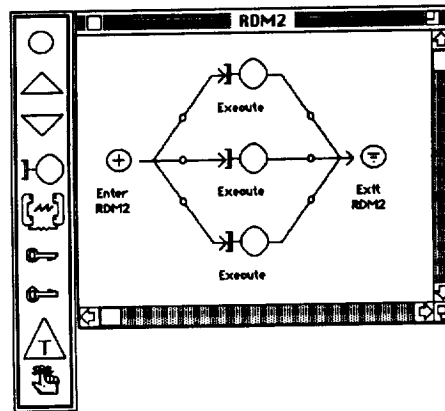
The Branching Probability Dialog
Figure 4.22



The Connector Statistics Dialog
Figure 4.23

4.1.2. RESOURCE DEMAND MODULES

Resource Demand Modules (RDMs) are a group of resource demands and the flow of control between those demands. Every module in an SCF window (previous Section) is associated with an RDM and that RDM defines the processing which occurs when that module is executed. In essence, an RDM is Pedestal™s version of program code.



An RDM Window
Figure 4.24

Figure 4.24 shows a simple RDM window which preforms three parallel execution requests. There are several icons in the palette of an RDM window. Each icon, except the first and last on the palette, represents a different type of resource demand:

Icon	Name	Purpose
△	Allocate	Increase the size of a data store by allocating more space on the data store's memory device.
▽	Deallocate	Decrease the size of a data store by deallocating some of its space on its memory device.
⊞	Execute	Execute instructions on a processor device.
⌋	Access	Access a data store. Accesses supported are random read, sequential read, insert, update, and reorganize.
🔑	Request Lock	Enter a protected region of code.
🔑	Relinquish Lock	Exit a protected region of code.
⚠	Delay	Delay for some time period.

The allocate, deallocate and access icons concern data stores. For now, think of data stores as files. The files reside on a memory device (disk). Allocate and deallocate change the size of the file by claiming more/less space on the disk the file resides on. The access icon allows the file contents to be modified. For more information regarding data stores and memory devices see Sections 4.4 and 4.5.2.2 respectively. In the RDM only the type of request and the data store is specified. Relating the data stores to a specific device is done via the mappings (Chapter 5).

The execute request causes a specified number of instructions to be executed. The actual device used, and thereby the time delay involved, is dependant on the task of the module invoking the RDM and the processor to which that task is mapped. For information about processors and mappings refer to Section 4.5.2.1 and Chapter 5 respectively.

The lock request and relinquish icons are used to bound special Sections of code. Refer to Section 4.3 for a complete description of locks.

The delay icon in an RDM is the same as a delay in an SCF. It is used to represent a delay caused by something not being modeled explicitly. Note that this delay occurs within a task; there can be no task boundaries within an RDM since the RDM is the smallest unit of partitioning.

Since the association of processors used for execution and the files used for data stores are determined by the mappings (Chapter 5) and not by the RDM diagram, **the RDM can be drawn independent of the hardware in the model.** The mappings allow these assignments to be changed easily and rapidly between simulations.

Input fusion and output fission can occur on all icons in the RDM window. The effect of fusion and fission is the same as that described for SCF windows (Section 4.1.1). One difference between RDM and SCF windows is that connectors in RDM windows only denote flow of control. Therefore, unlike in SCFs, transactions do not change size as they cross connectors within an RDM diagram. Also, there are no branching tables which give probabilities/rules for branching based on the workload which created the transaction. In RDMs, each connector has only one probability/rule, and the creator of the transaction plays no role in branching.

4.1.2.1. RDM WINDOWS

The object fields for an RDM window are set as follows:

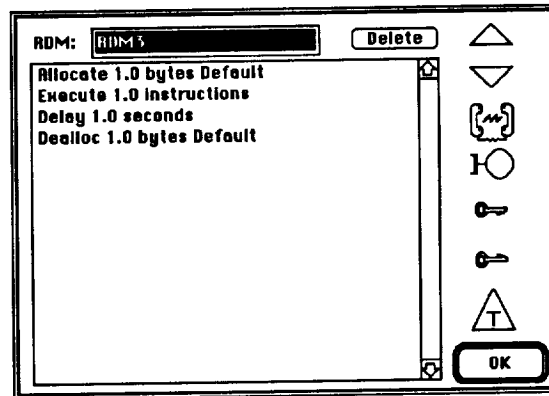
Field Name	Setting
WindowType	RDM
ConnectorType	HorizontalArrow {Varies}
CanEdit	True
CanGetInfo	True
CanMouse	True
CanReduce	True
CanTile	True

Connectors are also supported on the RDM window and the direction of the drawing can be changed with the "Toggle Vertical/Horizontal" item of the draw menu just like SCF windows. The method routines for an RDM window are:

Method Name	Function
ConnectorDoubleClick	Display the branching dialog (see below).
PlaceConnector	Creates a new control flow.
Activate	No function.

4.1.2.2. TEXT RDMS

An alternate to the window representation of the RDM is the textual form. A textual RDM is specified with the RDM Text dialog shown in Figure 4.25. The order of statements in the list determines the order of execution. Note that no parallelism is possible in textual RDMs.



The RDM Text Dialog
Figure 4.25

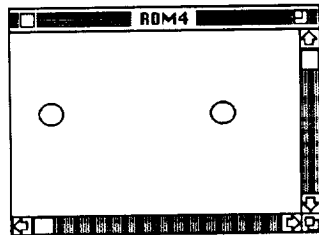
Clicking icons on the right side of the dialog will add a statement to the list. The statement will be appended to the end of the list if no statements in the list are highlighted when the icon is clicked. If a statement is highlighted when the icon is clicked, the new statement will be inserted in the list immediately before the highlighted statement. Double clicking one of the statements in the list will cause the dialog associated with that statement to be displayed, thus allowing the statement to be edited. See the following Sections concerning the individual icons for the content of these dialogs. Clicking on the "Delete" button while a statement is highlighted will cause that statement to be deleted.

Associated with each statement is a sentence dialog that is displayed in response to double clicking the statement. The sentence is of the general form: verb, amount, preposition, object. The preposition is a button that provides access to the definition of the object. This general concept will be particularized in the discussion of each icon and its related statement.

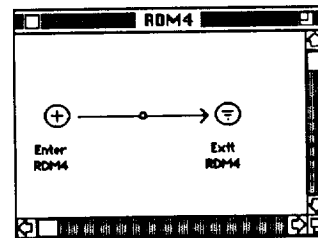
4.1.2.3.

THE STIMULATOR/TERMINATOR ICON

There must be one stimulus and one terminator on each RDM window. When a transaction arrives at a module it will be sent through the module's RDM starting at the stimulus. When the transaction reaches the terminator it will exit the module. To save space on the palette, one icon represents a generic form of these icons. When the user connects to/from the icon its species is changed to terminator/stimulus respectively. Figure 4.26 shows two generic icons placed in a window. Figure 4.27 shows the species of the stimulus and terminator after the icons are connected (the stimulus is on the left). Note that connectors may only be drawn to terminators, not from them. Similarly, connectors may only be drawn from a stimulus, not to it.



Generic Icons Before Connecting
Figure 4.26



Stimulus/Terminator Icons After
Connecting
Figure 4.27

This icon has its object fields set to:

Field Name	Setting
Genus	Stim/Term
Species	Stim or Term
Type	BlockIcon
Shadow	False

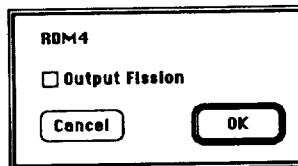
As mentioned above, the species of this icon changes to stimulus or terminator as soon as a connection is made to it. Shadowing is not used in RDM windows.

The methods for the stimulus/terminator icon are:

Method Name	Function
Deleted	Destroys the data structure associated with the icon.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	When a connection is made to a stimulus/terminator icon, the gender routine is activated and the species is set according

	to the direction of the connector. This routine also ensures that once the icon has its species, say stimulus, that connectors are only made to it in the right direction. I.E. stimulus icons may only be connected from not to.
Save	Saves the data structure associated with the icon to the model file.
Restore	Reads a data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	Only allows connectors to be drawn from stimulus icons and only allows connectors to be drawn to terminator icons.
DoubleClick	Double clicking a stimulus will cause the stimulus dialog be displayed (see below). Double clicking on a terminator is not allowed.
GetInfo	Same as double click (see above).

As mentioned in the method list, double clicking on a stimulus will display the stimulus dialog shown in Figure 4.28. The only thing to change on an RDM stimulus is whether or not output fission occurs. If the "Output Fission" check box is checked fission will occur. The default is for fission not to occur.



The Stimulus Dialog
Figure 4.28

4.1.2.4. THE ALLOCATE ICON

An allocate request is a request to increase the size of a data store. This is equivalent to increasing the number of sectors on a disk which are allocated for a file, or to increasing the size of the memory set aside for a dynamic data structure. For a discussion of data stores and the process of mapping data stores to memory devices see Sections 4.4 and 5.4 respectively.

The allocate icon has its object fields set to:

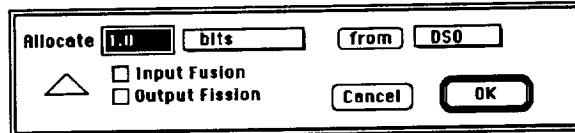
Field Name	Setting
Genus	Allocate
Species	Allocate
Type	BlockIcon
Shadow	False

Shadowing is not used in RDM windows. The methods for the allocate icon are:

Method Name	Function
Deleted	Destroys the demand data structure associated with the icon.
Duplicated	Attaches a demand data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated.
Pasted	Copies the demand in the Clipboard into the current window.
Gender	No function.
Save	Saves the request data structure associated with the icon to the model file.
Restore	Reads a request data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking an allocate icon will cause the allocation request dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on an allocate icon will display the allocation request dialog. Figure 4.29 shows a sample allocation request dialog.

The highlighted editable text field is where you specify the amount of the memory device to be allocated. This amount is specified as a number and a unit of memory. The "from" button will display the data store dialog (Section 4.4) for the data store displayed in the adjacent popup menu. That popup will contain the names of all currently defined data stores. Additionally, there is a "New..." entry which, if selected, will cause a new data store with a unique name to be created and picked as the data store for this request. The "Input Fusion" and "Output Fission" check boxes determine if fusion and/or fission are associated with this statement(action occurs if box is checked).



Allocation Request Dialog
Figure 4.29

The request dialog shown in Figure 4.29 shows the default settings for a new allocate icon.

4.1.2.5. THE DEALLOCATE ICON

A deallocate request is a request to decrease the size of a data store. This is equivalent to freeing some of the sectors on a disk which are allocated for a file, or to freeing some of the memory set aside for a dynamic data structure. For a discussion of data stores and the process of mapping data stores to memory devices see Sections 4.4 and 5.4 respectively.

The deallocate icon has its object fields set to:

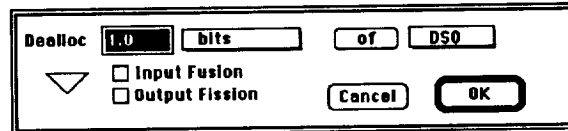
Field Name	Setting
Genus	Deallocate
Species	Deallocate
Type	BlockIcon
Shadow	False

Shadowing is not used in RDM windows. The methods for the deallocate icon are:

Method Name	Function
Deleted	Destroys the demand data structure associated with the icon.
Duplicated	Attaches a demand data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated.
Pasted	Copies the demand in the Clipboard into the current window.
Gender	No function.
Save	Saves the request data structure associated with the icon to the model file.
Restore	Reads a request data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking a deallocate icon will cause the deallocate request dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on a deallocate icon will display the deallocate request dialog. Figure 4.30 shows a sample deallocate request dialog.

The initially selected field is where you specify the amount of the memory device to be freed. This amount is specified as a number and a memory unit. The "of" button will display the data store dialog (Section 4.4) for the data store displayed in the adjacent popup menu. That popup will contain the names of all currently defined data stores. Additionally, there is a "New..." entry which, if selected, will cause a new data store with a unique name to be created and picked as the data store for this request. The "Input Fusion" and "Output Fission" check boxes determine if fusion and/or fission occur at this icon (action occurs if box is checked).



Deallocate Request Dialog
Figure 4.30

The request dialog shown in Figure 4.30 shows the default settings for a new deallocate icon.

4.1.2.6. THE EXECUTE ICON

An execute request is a request to have some instructions executed by a processor. The processor used will depend on the mapping of the task associated with the module which references the RDM. The object field therefore is not visible here as it is for the other types of demand. For a discussion of tasks, processors and the process of mapping tasks to processor devices see Sections 4.2, 4.5.2.1 and 5.3 respectively.

The execute icon has its object fields set to:

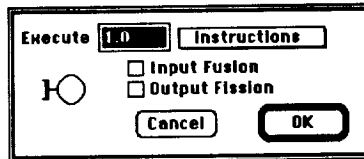
Field Name	Setting
Genus	Execute
Species	Execute
Type	BlockIcon
Shadow	False

Shadowing is not used in RDM windows. The methods for the Execute icon are:

Method Name	Function
Deleted	Destroys the demand data structure associated with the icon.
Duplicated	Attaches a demand data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated.
Pasted	Copies the demand in the Clipboard into the current window.
Gender	No function.
Save	Saves the request data structure associated with the icon to the model file.
Restore	Reads a request data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking an execute icon will cause the execute request dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on an execute icon will display the execute request dialog. Figure 4.31 shows a sample execute request dialog.

The highlighted field is where you specify the number of instructions to be executed. This amount is specified as a number and a unit (powers of 10). The "Input Fusion" and "Output Fission" check boxes determine if fusion and/or fission are associated with this statement (action occurs if box is checked).



Allocation Request Dialog
Figure 4.31

The request dialog shown in Figure 4.31 shows the default settings for a new execute icon.

4.1.2.7. THE ACCESS ICON

An access request is a request to access a data store. This will cause messages to be sent between the processor the transaction is executing on and the memory device the data store resides on. The types of access supported are: random read, sequential read, insert, update, and reorganize. These are easily understood as file transactions, but apply just as well to memory management. For a discussion of hardware and mappings see Section 4.5 and Chapter 5 respectively.

The access icon has its object fields set to:

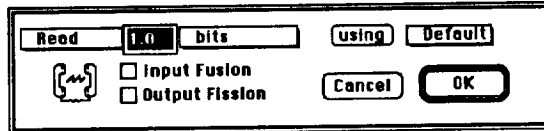
Field Name	Setting
Genus	Access
Species	Access
Type	BlockIcon
Shadow	False

Shadowing is not used in RDM windows. The methods for the access icon are:

Method Name	Function
Deleted	Destroys the demand data structure associated with the icon.
Duplicated	Attaches a demand data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated.
Pasted	Copies the demand in the Clipboard into the current window.
Gender	No function.
Save	Saves the request data structure associated with the icon to the model file.
Restore	Reads a request data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking an access icon will cause the access request dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on an access icon will display the access request dialog. Figure 4.32 shows a sample access request dialog.

At the top left of the dialog is a popup (currently showing "Read"), this popup is used to specify the type of access being performed. The amount of data involved is specified by the next two fields. This amount is specified as a number and a scale (memory units). The "using" button will display the data store dialog (Section 4.4) for the data store displayed in the adjacent popup menu. That popup will contain the names of all currently defined data stores. Additionally, there is a "New..." entry which, if selected, will cause a new data store with a unique name to be created and picked as the data store for this request. The "Input Fusion" and "Output Fission" check boxes determine if fusion and/or fission occur at this icon (action occurs if box is checked).



Access Request Dialog
Figure 4.32

The request dialog shown in Figure 4.32 shows the default settings for a new access icon.



4.1.2.8. THE REQUEST LOCK ICON

A lock request is a request to enter a controlled Section of code. For a discussion of locks (locked regions of code) see Section 4.3.

The request lock icon has its object fields set to:

Field Name	Setting
Genus	RequestLock
Species	RequestLock
Type	BlockIcon
Shadow	False

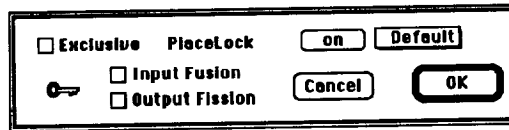
Shadowing is not used in RDM windows. The methods for the request lock icon are:

Method Name	Function
Deleted	Destroys the demand data structure associated with the icon.
Duplicated	Attaches a demand data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated.
Pasted	Copies the demand in the Clipboard into the current window.
Gender	No function.
Save	Saves the request data structure associated with the icon to the model file.
Restore	Reads a request data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking a request lock icon will cause the lock request dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on a request lock icon will display the lock request dialog. Figure 4.33 shows a sample lock request dialog.

"Exclusive" is a check box which determines if this entry to the protected region of code is exclusive (if checked it is exclusive). The "on" button will display the lock dialog (Section 4.3) for the lock displayed in the adjacent popup menu. That popup will contain the names of all currently defined locks.

Additionally, there is a "New..." entry which, if selected, will cause a new lock with a unique name to be created and picked as the lock for this request. The "Input Fusion" and "Output Fission" check boxes determine if fusion and/or fission occur at this icon (action occurs if box is checked).



Allocation Request Dialog
Figure 4.33

The request dialog shown in Figure 4.33 shows the default settings for a new request lock icon.

4.1.2.9. THE RELINQUISH LOCK ICON

A relinquish lock request is a notification that the protected Section of code is being exited. For a discussion of locks see Section 4.3.

The allocate icon has its object fields set to:

Field Name	Setting
Genus	RelinquishLock
Species	RelinquishLock
Type	BlockIcon
Shadow	False

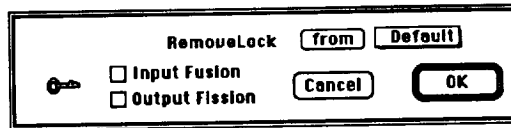
Shadowing is not used in RDM windows. The methods for the relinquish lock icon are:

Method Name	Function
Deleted	Destroys the demand data structure associated with the icon.
Duplicated	Attaches a demand data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated.
Pasted	Copies the demand in the Clipboard into the current window.
Gender	No function.
Save	Saves the request data structure associated with the icon to the model file.
Restore	Reads a request data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking a relinquish lock icon will cause the relinquish lock request dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on a relinquish lock icon will display the relinquish lock request dialog. Figure 4.34 shows a sample relinquish lock request dialog.

The "from" button will display the lock dialog (Section 4.3) for the lock displayed in the adjacent popup menu. That popup will contain the names of all currently defined locks. Additionally, there is a "New..." entry which, if selected, will cause a new lock with a unique name to be created and picked as the lock

for this request. The "Input Fusion" and "Output Fission" check boxes determine if fusion and/or fission occur at this icon (action occurs if box is checked).



Allocation Request Dialog
Figure 4.34

The request dialog shown in Figure 4.34 shows the default settings for a new relinquish lock icon.

4.1.2.10. THE DELAY ICON

The delay icon is used to represent delays caused in the system by events not being explicitly simulated in Pedestal™. The delay icon has its object fields set to:

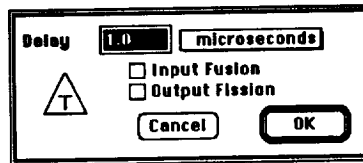
Field Name	Setting
Genus	Delay
Species	Delay
Type	BlockIcon
Shadow	False

There is only one species of the Delay icon. Shadowing is not used in RDM windows. The methods for the delay icon are:

Method Name	Function
Deleted	Destroys the demand data structure associated with the icon.
Duplicated	Attaches a demand data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated.
Pasted	Copies the demand in the Clipboard into the current window.
Gender	No function.
Save	Saves the request data structure associated with the icon to the model file.
Restore	Reads a request data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	No restrictions are imposed.
DoubleClick	Double clicking a delay icon will cause the delay request dialog (see below) to be displayed.
GetInfo	Same as DoubleClick (see above).

As mentioned in the method list, double clicking on a delay icon will display the delay request dialog. Figure 4.35 shows a sample delay request dialog.

The length of the delay is determined by the "Delay" field. The time unit popup provides the scale of the delay. It should be noted that the use of expressions allows the length of the delay to change over time. The "Input Fusion" and "Output Fission" check boxes denote the absence (if unchecked) or presence (if checked) of the associated operations on this icon.



Delay Request Dialog
Figure 4.35

The request dialog shown in Figure 4.35 shows the default settings for a new delay icon.

4.1.2.11. RDM CONNECTORS

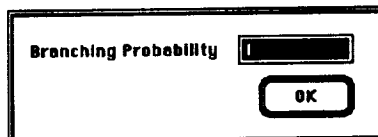
The connectors on an RDM window are used to denote existence and direction of control flow paths. These connectors determine the execution of resource demands throughout the RDM. The note icon has nothing to do with the simulation of software and therefore can not have data flows to or from it. The stimulus icon can only have connectors drawn from it and the terminator can only have connectors drawn to them. All the request icons can (and must) have connectors both to and from them.

The HotSpotIcon, Flag1Icon and Flag2Icon areas of RDM connectors are not used. RDM connectors always appear as directed lines. The object field settings for connectors are:

Field Name	Setting
Flag1Icon	None
Flag2Icon	None
HotSpotIcon	None

The method routines for a connector are:

Method Name	Function
Deleted	Destroys the branching probability associated with the connector.
Duplicated	Attaches a branching probability to the new duplicate connector. The probability will be the same as the original connector.
Pasted	If the icons on both ends of the connector are also pasted then the connector is pasted.
Gender	No function.
Save	Saves the branching probability associated with the connector to the model file.
Restore	Reads a branching probability from the model file and recreates the structure. This structure will then be associated with the proper connector.
DoubleClick	Double clicking a connector will display the branching dialog (see below).



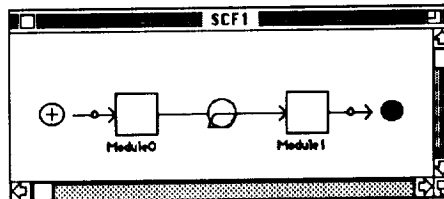
The Branching Dialog
Figure 4.36

As mentioned in the method list, double clicking on a connector will display the branching dialog. This dialog, shown in Figure 4.36 allows the branching probability associated with the connector to be viewed/edited. The default probability is 1. It should be noted that with the use of expressions, branching may be rule based instead of strictly probabilistic.

4.2. TASKS

One of the major factors which determines the performance of software on a multiprocessor time critical system is the way in which the software modules are grouped and assigned to processors. Pedestal™ calls these groups of modules tasks. Software modules are assigned to tasks either from the module to task mapping (see Section 5.2) or from the module dialog (see Section 4.1.1.3). Tasks are then assigned to processors through the task to processor mapping (see Section 5.3). Tasks have several characteristics which may be modified to determine the behavior of the task.

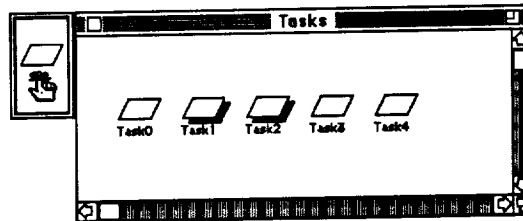
Figure 4.37 shows a sample SCF with two modules. The modules are in separate tasks as indicated by the ITC modifier (Section 4.1.1) to the connector between the modules. The function of tasks will be demonstrated by explaining what happens when a transaction finishes processing in Module0 and is sent to Module1. Assume that Module0 is in Task0 and Module1 is in Task1. Before the transaction can be passed to Module1, Task1 must be able to accept another transaction. A task has a fixed number of instantiations which it can handle. This number can be thought of as the maximum number of transactions which can be in the task at any given time. If the current number of instantiations of Task1 is less than its maximum, the transaction will exit Task0, be sent to the Task1 processor, enter Task1 and be processed by Module1. If the current number of instantiations of Task1 equals its maximum, the buffer must be checked. A task has a buffer which stores entry requests. If the current size of the Task1 buffer is less than its max, the transaction will exit Task0, be sent to the Task1 processor and be queued in the entry request buffer of Task1. If the Task1 buffer is full, then the transaction can not yet leave Task0 and must be queued until a transaction is removed from the Task1 buffer.



A Simple SCF
Figure 4.37

4.2.1. THE TASK WINDOW

The task window, shown in Figure 4.38, has an icon for every task which is currently defined and provides a convenient method to edit and/or create tasks. The task window palette also provides the note icon.



The Task Window
Figure 4.38

The object fields for the task window are set as follows:

Field Name	Setting
WindowType	Task
ConnectorType	NoConnector
CanEdit	True
CanGetInfo	True
CanMouse	True
CanReduce	True
CanTile	True

The NoConnector connector type denotes that connectors are not used on the task window. The task window methods are:

Method Name	Function
ConnectorDoubleClick	Not allowed.
PlaceConnector	Not allowed.
Activate	No function.

4.2.2. THE TASK ICON

The task icon allows the user to add and/or edit tasks. This icon has its object fields set to:

Field Name	Setting
Genus	Task
Species	Task
Type	BlockIcon
Shadow	False {Variable}

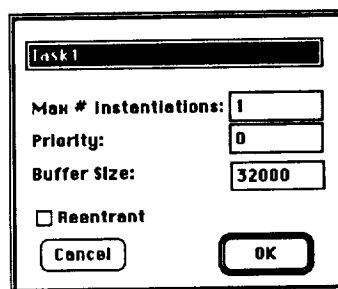
The shadow field is set to true if the number of instantiations of the task is greater than one. The methods for the task icon are:

Method Name	Function
Deleted	Delete the task associated with the icon and remove all references to this task by modules.
Duplicated	Create a new task and associate it with the new duplicate icon. The characteristics of the new task will be the same as the task being duplicated, only the name will change. The task names must be distinct.
Pasted	Places a previously duplicated or cut task icon on the window.
Gender	No function.
Save	Saves the task information associated with the icon to the model file.
Restore	Reads a task definition from the model file and associates it with the proper window and icon.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the task dialog to be displayed (see below).
GetInfo	Same as double click.

As mentioned in the method list, double clicking on a task icon will display the task dialog shown in Figure 4.39. This dialog may also be reached by clicking on the "Task" button on the module dialog (see Section 4.1.1.3). The first field on the task dialog allows the name of the task to be changed. All modules which reference the task will still reference it after the name is changed. The maximum number of instantiations determines the number of modules in the task which can be executing at the same time. The priority is the priority of the task and may affect the order in which resource requests made by modules within the task are processed. The Buffer Size determines the number

of messages (requests for the activation of a module in the task) which can be queued. If the number of instantiations is greater than one then the reentrant flag determines if an activation of the task when the task is currently active will suffer a task initialization delay.

The default value for instantiations, priority, buffer size and reentrant are shown below in Figure 4.39.



The Task Dialog

A screenshot of a dialog box titled "Task 1". It contains four input fields: "Max # Instantiations" with the value "1", "Priority" with the value "0", "Buffer Size" with the value "32000", and a checkbox labeled "Reentrant" which is currently unchecked. At the bottom of the dialog are two buttons: "Cancel" and "OK".

Task 1	
Max # Instantiations:	1
Priority:	0
Buffer Size:	32000
<input type="checkbox"/> Reentrant	
Cancel	OK

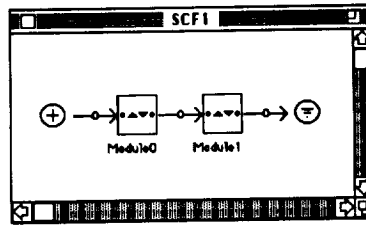
The Task Dialog
Figure 4.39

4.3. LOCKS

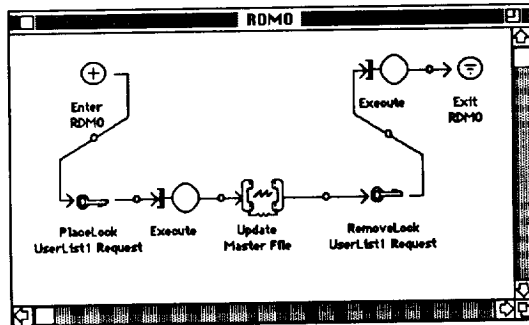
The flow of transactions may be controlled by placing restrictions on the number of transactions currently executing a portion of code. The concept is similar to that of task instantiation presented in Section 4.2; locks, however, may span task boundaries. Pedestal™ supports this concept with the placement of a place lock request at the beginning of the restricted code segment and a remove lock request at the point where the restriction is removed. The request will specify the logical name of the lock so that regions of code in different RDMs may use the same lock.

A request for a lock may be either exclusive or non-exclusive. Since the request and not the lock itself determines how many transactions may be within a restricted segment, delay may be introduced while an exclusive request is waiting for transactions to free a lock as well as while the exclusive hold is in effect. When an exclusive request arrives at a lock there may be several non-exclusive requests currently being service. The exclusive request is queued until there are no transactions currently using the lock; it then seizes the lock and blocks all other transactions from entering the lock until it issues its remove lock request. Non exclusive requests and remove requests are counted without association with a particular transaction. The exclusive request must however remain associated with the transaction since only the transaction holding the lock may release it.

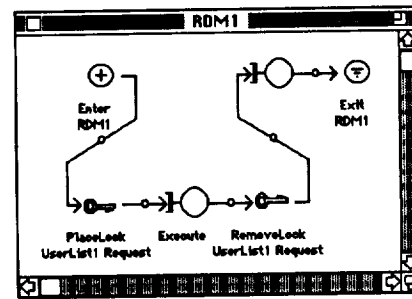
Figures 4.40, 4.41 and 4.42 show a simple SCF with two modules and the RDMs associated with each module. Module0 uses RDM0 and Module1 uses RDM1. Assume that there is a data structure called UserList1 which is not being modeled in Pedestal™. That is, the process of accessing the data and modeling the memory device in which it would reside are not important enough to model explicitly. Module0 references UserList1 and does some processing and a data access (to something else) based on the reference, then does some more processing. Module1 wants to update UserList1. To keep transactions from entering the part of Module0 that references UserList1 while UserList1 is being updated in Module1, a lock must be used. RDM0 shows the resource demands for request and relinquish lock requests using a lock named "UserList1 Reference". The request in this RDM is nonexclusive since UserList1 is only referenced, not changed. RDM1 also has the proper resource requests surrounded by a lock request and relinquish for the UserList1 lock. This request is exclusive. The effect of this is that when a transaction arrives at the lock request in RDM1, the transaction will be queued. The lock request in RDM0 will now queue any transactions that arrive, thus not letting them past the lock request. As soon as the last transaction that was in the protected region of RDM0 reaches the lock relinquish in RDM0, the transaction waiting at the lock request in RDM1 will be dequeued and allowed to execute. Until this transaction arrives at the lock relinquish in RDM1, no transaction will be allowed past the lock requests in either RDM0 or RDM1.



A Simple SCF
Figure 4.40



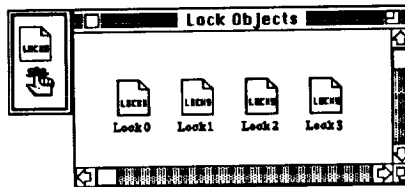
The RDM for Module0
Figure 4.41



The RDM for Module1
Figure 4.42

4.3.1. THE LOCK WINDOW

The lock window, shown in Figure 4.43, has an icon for every logical lock which is currently defined and provides a convenient method to edit and/or create new locks. The lock window palette also supplies the note icon. See Section 4.1.1.5 for a complete description of the note icon.



The Lock Window
Figure 4.43

The object fields for the lock window are set as follows:

Field Name	Setting
WindowType	Lock
ConnectorType	NoConnector
CanEdit	True
CanGetInfo	True
CanMouse	True
CanReduce	True
CanTile	True

The lock window methods are:

Method Name	Function
ConnectorDoubleClick	Not allowed.
PlaceConnector	Not allowed.
Activate	No function.



4.3.2. THE LOCK ICON

The lock icon allows the user to add and/or edit locks. This icon has its object fields set to:

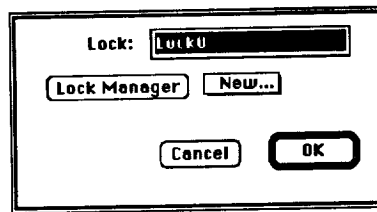
Field Name	Setting
Genus	Lock
Species	Lock
Type	BlockIcon
Shadow	False

The methods for the lock icon are:

Method Name	Function
Deleted	Delete the lock associated with the icon and remove all references to this lock by request and relinquish lock requests.
Duplicated	Create a new lock and associate it with the new duplicate icon. The characteristics of the new lock will be the same as the lock being duplicated, only the name will change. The lock names must be distinct.
Pasted	Places a previously duplicated or cut lock icon on the window.
Gender	No function.
Save	Saves the lock information associated with the icon to the model file.
Restore	Reads a lock definition from the model file and associates it with the proper window and icon.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the lock dialog to be displayed (see below).
GetInfo	Same as double click.

As mentioned in the method list, double clicking on a lock icon will display the lock dialog shown in Figure 4.44. This dialog may also be reached by clicking on the "on" button on the place and remove lock dialogs (see Sections 4.1.2.8 and 4.1.2.9). The exclusive/nonexclusive aspect of a lock is denoted on the place lock request. The lock name may be changed, but all lock names must be distinct. The "Lock Manager" button will display the lock manager of the operating system associated with this lock. This operating system will be displayed in the adjacent popup menu. This menu will contain the names of all currently defined operating systems. Additionally, there will be a "New..." entry, clicking on this entry will cause a new operating system to be

created with a unique name and will associate the new operating system to this lock. For more information about operating systems and managers refer to Section 4.6.



The Lock Dialog
Figure 4.44

The lock dialog shown in Figure 4.44 displays the default contents of the dialog for a newly created lock.

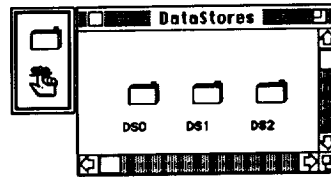
4.4. DATASTORES

The purpose of a program is to process data. Data may be organized in various ways such as data structures in memory or files, etc.. Pedestal™ uses datastores to represent groupings of data. Requests to access a datastore in the software are made in the RDM (Section 4.1.2). The requests specify the type of access (read, update, etc.), the amount of data involved (fixed record size within a datastore is not assumed), and the name of the datastore to access. The characteristics of the datastore determine the speed of the various access types, thereby simulating the structure of the data. The datastore to memory mapping (Section 5.4) is used to choose the device on which a datastore resides.

Memory devices (used for all types of storage devices, Section 4.5.2.2) can be accessed either sequentially or randomly. The rate at which a particular memory device can be accessed in the two modes determines the type of "real" storage device being simulated. A datastore represents the structure of a data set by defining the number of random and sequential accesses necessary for each type of data access.

4.4.1. THE DATASTORE WINDOW

The datastore window, shown in Figure 4.45, has an icon for every datastore which is currently defined and provides a convenient method to edit and/or create new locks. The lock window palette also supplies the note icon which is described in Section 4.1.1.5.



The Datastore Window
Figure 4.45

The object fields for the datastore window are set as follows:

Field Name	Setting
WindowType	Datastore
ConnectorType	NoConnector
CanEdit	True
CanGetInfo	True
CanMouse	True
CanReduce	True
CanTile	True

The datastore window methods are:

Method Name	Function
ConnectorDoubleClick	Not allowed.
PlaceConnector	Not allowed.
Activate	No function.

4.4.2. THE DATASTORE ICON

The datastore icon allows the user to add and/or edit datastores. This icon has its object fields set to:

Field Name	Setting
Genus	Datastore
Species	Datastore
Type	BlockIcon
Shadow	False

The methods for the datastore icon are:

Method Name	Function
Deleted	Delete the datastore associated with the icon and remove all references to this datastore by data access, allocate and deallocate requests.
Duplicated	Create a new datastore and associate it with the new duplicate icon. The characteristics of the new datastore will be the same as the lock being duplicated, only the name will change. The datastore names must be distinct.
Pasted	Places a previously duplicated or cut datastore icon on the window.
Gender	No function.
Save	Saves the datastore information associated with the icon to the model file.
Restore	Reads a datastore definition from the model file and associates it with the proper window and icon.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the datastore dialog to be displayed (see below).
GetInfo	Same as double click.

As mentioned in the method list, double clicking on a datastore icon will display the datastore dialog shown in Figure 4.46. This dialog may also be reached by clicking on the "from" button on the data access, allocate and deallocate request dialogs (see Section 4.1.2). The datastore name may be changed in the "Datastore" field. However, all datastore names must be distinct. The size of the datastore, when a simulation begins, is defined as a number and a unit (provided by the popup menu). The array of number fields are used to

specify the number of sequential and nonsequential accesses which must be made to the memory device the datastore resides on to perform each type of data access.

The Datastore Dialog box contains the following fields and controls:

- Datastore:** A text field containing the value "USU".
- Size:** A text field containing the value "1" and a unit dropdown menu set to "kilobits".
- # Accesses required to perform a:** A section with two columns: "(Sequential)" and "(Random)".
- Read:** Input fields for Sequential (0) and Random (0).
- Read_Seq:** Input fields for Sequential (0) and Random (0).
- Insert:** Input fields for Sequential (0) and Random (0).
- Update:** Input fields for Sequential (0) and Random (0).
- Reorganize:** Input fields for Sequential (0) and Random (0).
- Buttons:** "Cancel" and "OK" buttons at the bottom.

The Datastore Dialog
Figure 4.46

The datastore dialog shown in Figure 4.46 displays the default contents of a newly created datastore.

4.5. HARDWARE

In Pedestal™, hardware is defined as a hierarchical group of nodes. Each node is composed of processor devices, memory devices, bus devices, and communication links between the devices. The topmost node in the hierarchy is the hardware window called "HW". Each processor, memory or bus has parameters which define its speed of operation and capacity. Additionally, every device may have an operating system associated with it. The operating system determines how the queues are managed for a device and what the overhead costs are. Operating systems can be associated with devices either directly from the device dialogs or from the device to manager mapping. See Section 4.6 and Chapter 5 for discussions of operating systems and the device to manager mapping.

Communication paths between devices are represented by connectors on hardware windows. Since there may be more than one communication path between any two devices, a routing scheme is necessary. The scheme adopted by Pedestal™ is to put a routing set on each connector. The routing set contains a routing list for each icon attached to the connector. The list contains devices reachable from the associated device and the probability that this communication path will be used to send messages from the device to the listed device. Note that the use of expressions (Chapter 3) allows for the routing to be more sophisticated than simple probabilistic routing. Processor and memory devices appear in routing lists along with the name of all hardware windows except the window containing the specific device; buses are not a valid destination for a message.

Messages are generated primarily for data accesses, allocations and deallocations (Section 4.1.2). Messages are also generated when a transaction crosses a task boundary and the two tasks involved reside on different processors (Section 4.2). For example, if a process wishes to update a datastore, a message must be sent from the processor the task is executing on to the memory the datastore resides in. The message will be sent from the processor to the memory device by walking the connectors based on the routing list entries for the memory device.

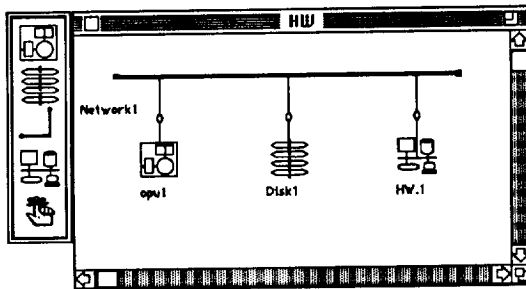
In the example just used, if the devices (the processor and the memory) reside in different windows in the hardware hierarchy, the message will first be routed based on the name of the node (window) the memory device is in. The message will be sent from the processor to a boundary or node icon in the same window as the memory. Next, the message will be routed from the boundary/node icon to the memory device. Since the message is now in the proper window, routing proceeds as dictated by the routing list entries for the memory device.

4.5.1. THE HARDWARE & NODE WINDOWS

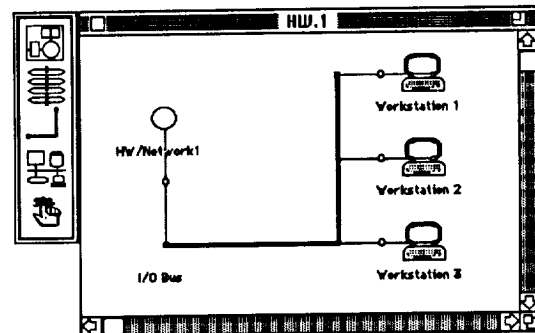
The hardware window, shown in Figure 4.47, is where the hardware architecture of the system being modeled is defined. The first three icons in the hardware window palette correspond to the primitive hardware devices which Pedestal™ supplies. These icons represent processors, memories and communication buses.

The next icon in the palette, called a node icon, is a hook to another hardware window, thus allowing the architecture to be defined hierarchically in a top down fashion. Figure 4.48 shows the window associated with the node icon in the top level hardware window shown in Figure 4.47 (the icons in this Figure labeled workstations are an alternate form of the processor icon). The window associated with the node icon will have a boundary icon (not in the palette) placed on it for each connector drawn into the node. The name of the boundary icon will provide the window and icon name of the icon on the other end of that specific connector to the node icon.

The last icon in the palette is the note icon which allows user documentation to be added to the drawing. Refer to Section 4.1.1.5 for a description of the note icon. More specific descriptions of the icons in the hardware windows will be given in the proper Sections below.



A Hardware Window
Figure 4.47



A Hardware Window With a Boundary
Icon
Figure 4.48

The object fields for the hardware window are set as follows:

Field Name	Setting
WindowType	Hardware
ConnectorType	NoArrow
CanEdit	True
CanGetInfo	True
CanMouse	True
CanReduce	True
CanTile	True

The NoArrow connector type specifies that connectors are supported, but they are undirected and thus are drawn as a line with no arrow on the end. Connectors are also supported on the hardware window, thus some of the window methods are used:

Method Name	Function
ConnectorDoubleClick	Display the first level routing dialog (see below).
PlaceConnector	Add proper device names to the new connectors routing table (see below).
Activate	none

4.5.2. HARDWARE DEVICES

A device icon represents a pool of devices. The definition of the icon consists of the following information: performance information, a manager which handles its queue and overhead information, and the number of devices in the pool. The performance information includes things such as the execution rate of a processor, the size and access speeds of a memory, etc.. This specific information is defined in device type dialogs.

There is a set of device types for each device class supported by Pedestal™: processors, memories and busses. These "types" contain the performance information part of a device definition. The types can be shared by icons of the proper device class; i.e., multiple processor icons can reference the same processor "type".

A device manager determines how queues are managed and defines the operating system overheads associated with that class of device. These managers can also be shared by icons of the proper class; multiple processor icons can reference the same processor manager, regardless of the "type" of the icons. For more information concerning managers see Section 4.6.

The user is allowed to change the icon used to represent a processor or memory icon by clicking on the icon shown in the device dialog, which is accessed by double clicking on the window icon. Since a memory device may be simulating a memory or a tape or a disk, depending on the performance characteristics, the specific icon can be changed to one which depicts the type of device the user intends to simulate. Similarly, icon species are provided for processors to represent a processor, workstation and a network bridge.

4.5.2.1. THE PROCESSOR ICON

The processor icon on the Hardware window allows the user to define a processor pool. A processor pool is a group of one or more processors with the same characteristics which share a common task queue. This icon has its object fields set to:

Field Name	Setting
Genus	Processor
Species	Processor {Variable}
Type	BlockIcon
Shadow	False {Variable}

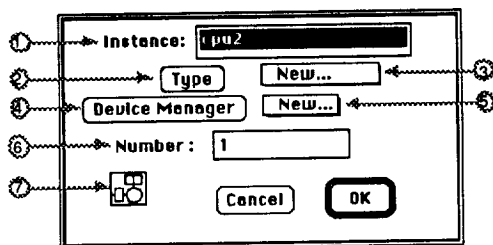
The shadow field is initially false on a processor icon since the default number of processing elements in the pool is 1. If the number of processing elements is changed to more than one, the shadow field is set to true and the icon will be drawn with a shadow. The species, and thus the on screen representation, of the processor pool can be changed by the user from the device dialog that is shown when the window icon is double clicked.

The methods for the processor icon are:

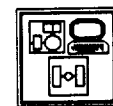
Method Name	Function
Deleted	Destroys the device data structure associated with the icon.
Duplicated	Attaches a device data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated. However, the name will be changed so that it is unique from all other names in the window.
Pasted	Assures that the name of the device data structure associated with the icon being pasted is distinct from all other devices in the window being pasted into. The name will be changed if necessary.
Gender	Ensures that the name of the icon's device data structure is on the window's routing tables if there are any connectors to the icon.
Save	Saves the device data structure associated with the icon to the model file.
Restore	Reads a device data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.

Connected	Allows connectors to be drawn to and from the icon.
DoubleClick	Double clicking this icon will cause the device data structure to be created for this icon if it does not yet exist. Additionally, the device dialog will be displayed for the structure (see below).
GetInfo	Selecting this icon and choosing the "Get Info" item of the Edit menu will cause the device data structure to be created for this icon if it does not yet exist. Additionally, the devices type dialog will be displayed for the structure (see below).

As mentioned in the method list, double clicking on a processor icon will display the processor pool dialog shown in Figure 4.49. This dialog contains the information which defines the processor pool. The name (field 1), in this case "cpu2", is the name the icon will be given. This name must be distinct from all other device names in the same window. The processor type popup menu (field 3) will contain a list of all defined processor types with the processor type of this pool showing. A processors type defines the execution characteristics of the pool. Clicking on the "Type" button (field 2) will display the processor type dialog explained below. The popup menu next to the "Device Manager" button (field 5) will contain a list of all defined device managers with the manager of this pool showing. Clicking on the "Device Manager" button (field 4) will display the device manager dialog. Managers are explained in Section 4.6. The number field (field 6) shows the number of processors in this pool. Finally, field 7 is a button. Clicking on this button will display a dialog, shown in Figure 4.50, which allows the species (icon used to represent this pool) of the processor icon to be changed. Just click on the preferred icon and the change will be made. The intended use of the species are processor, workstation and network bridge.



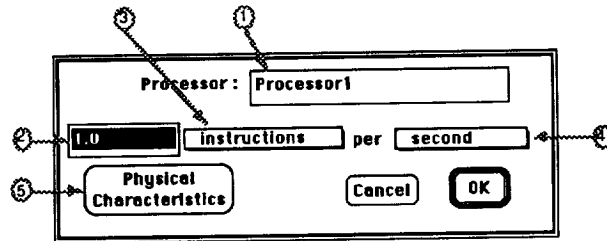
The Processor Pool Dialog
Figure 4.49



The Processor Species Dialog
Figure 4.50

The processor type dialog shown in Figure 4.51 is accessed by clicking on the "Type" button of the processor pool dialog or by selecting a processor icon and selecting the "Get Info" item of the Edit menu. A processor type determines the execution rate of a processor. A given type may be used by any number of processor pools. That is, each processor icon represents a unique named processor pool, but these pools may be of the same type. The

processor type name (field 1) must be distinct from all other processor types. The base execution rate is defined by fields 2, 3 & 4.



The Processor Type Dialog
Figure 4.51

The last field on the processor type dialog is the physical characteristics button (field 5). Clicking this button will display the physical characteristics dialog, shown in Figure 4.52, for this processor type. This is provided solely for use as documentation of the system being modeled. The "Loading" item of the run menu (Section 2.7) provides the totals of these values for all devices in a model.

Physical Characteristics		
Size	<input style="width: 90%;" type="text" value="0"/>	cu.ft.
Weight	<input style="width: 90%;" type="text" value="0"/>	lbs.
Power	<input style="width: 90%;" type="text" value="0"/>	watts
MTBF	<input style="width: 90%;" type="text" value="0"/>	hours
MTTR	<input style="width: 90%;" type="text" value="0"/>	hours
<input style="width: 40%;" type="button" value="Cancel"/> <input style="width: 40%;" type="button" value="OK"/>		

The Physical Characteristics Dialog
Figure 4.52

4.5.2.2. THE MEMORY ICON

The memory icon on the Hardware window allows the user to define a memory pool. A memory pool is a group of one or more memories with the same characteristics which share a common memory request queue. As implied by the memory icon, a memory in Pedestal™ is a data storage device. This storage device may be a memory, disk or tape. The characteristics determine which physical device type is actually being simulated. The memory icon has its object fields set to:

Field Name	Setting
Genus	Memory
Species	Disk {Variable}
Type	BlockIcon
Shadow	False {Variable}

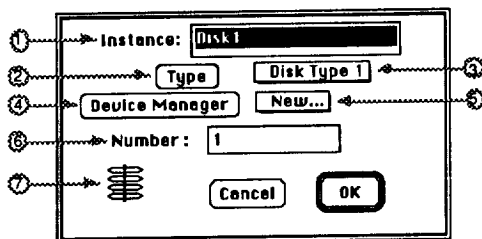
The shadow field is initially false on a memory icon since the default number of memory devices in the pool is 1. If the number of memory devices is changed to more than one, the shadow field is set to true and the icon will be drawn with a shadow. The species of the memory icon can be changed also, allowing the icon which is drawn on the window to appear as either a disk, tape or memory. The appearance (species) of the icon is only for notational purposes, it does not mean that the characteristics of the pool actually define such a device. Making the characteristics match the drawn icon is the user's job.

The methods for the memory icon are:

Method Name	Function
Deleted	Destroys the device data structure associated with the icon.
Duplicated	Attaches a device data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated. However, the name will be changed so that it is unique from all other names in the window.
Pasted	Ensures that the name of the device data structure associated with the icon being pasted is distinct from all other devices in the destination window. The name will be changed if necessary.
Gender	Ensures that the name of the icon's device data structure is on the window's routing tables if there are any connectors to the icon.

Save	Saves the device data structure associated with the icon to the model file.
Restore	Reads a device data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	Allows connectors to be drawn to and from the icon.
DoubleClick	Double clicking this icon will cause the device data structure to be created for this icon if it does not yet exist. Additionally, the device dialog will be displayed for the structure (see below).
GetInfo	Selecting this icon and choosing the "Get Info" item of the Edit menu will cause the device data structure to be created for this icon if it does not yet exist. Additionally, the device type dialog will be displayed for the structure (see below).

As mentioned in the method list, double clicking on a memory icon will display the memory device pool shown in Figure 4.53. This dialog contains the information which defines the memory pool. The name (field 1), in this case "Disk1", is the name the icon will be given. This name must be distinct from all other device names in the same window. The memory type popup menu (field 3) will contain a list of all defined memory types with the memory type of this pool showing. A memory pool type defines the transmission and storage characteristics. Clicking on the "Type" button (field 2) will display the memory type dialog explained below. The popup menu next to the "Device Manager" button (field 5) will contain a list of all defined device managers with the manager of this pool showing. Clicking on the "Device Manager" button (field 4) will display the device manager dialog. Managers are explained in Section 4.6. The number field (field 6) shows the number of memory elements in this pool. Finally, the icon (field 7) allows the iconic representation of the pool to be changed. Clicking on the icon will display the memory icon dialog shown in Figure 4.54. The user picks the iconic representation by clicking on one of the icons in the memory icon dialog.

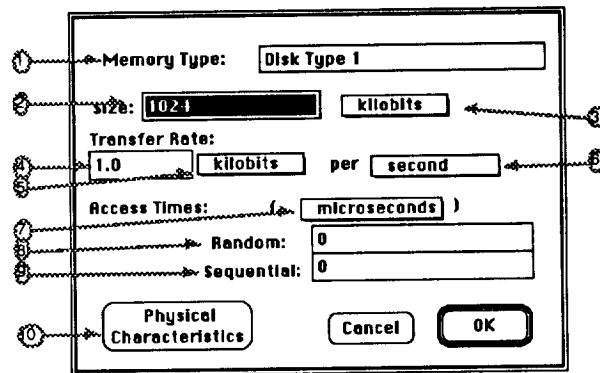


The Memory Pool Dialog
Figure 4.53



The Memory Icon Dialog
Figure 4.54

The memory type dialog shown in Figure 4.55 is accessed by clicking on the "Type" button of the memory pool dialog or by selecting a memory icon and choosing the "Get Info" item of the Edit menu. A memory type determines the size, access time and transmission rate of a memory pool. A given type may be used by any number of memory pools. That is, each memory icon represents a unique named memory pool, but these pools may be of the same type. The memory name (field 1) must be distinct from all other memories. Memory element capacity is determined by the size and unit popup menu (fields 2 & 3). The transfer rate is defined by fields 4, 5 & 6. Fields 5 & 6 are popup menus. Fields 8 & 9 define the time required to make one access to the memory depending on the access type. The time unit menu (field 7) applies both to the random and sequential access time (fields 8 & 9). Lastly, the "Physical Characteristics" button (field 10) is clicked to display the physical characteristic dialog for this type of memory. The physical characteristics dialog operates the same as the one described in Section 4.5.2.1 concerning the processor icon.



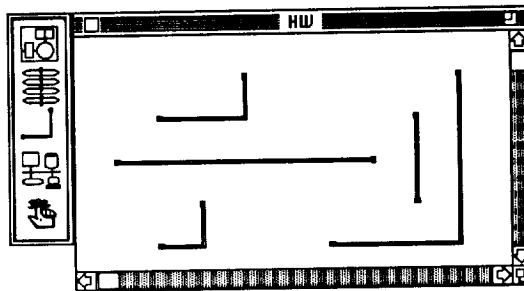
The Memory Type Dialog
Figure 4.55

4.5.2.3. THE BUS ICON

The bus icon on the Hardware window allows the user to define a pool of communications buses. The pool is a group of communication channels with the same transmission characteristics who share a common transmission request queue. This icon has its object fields set to:

Field Name	Setting
Genus	Bus
Species	Bus
Type	LineIcon
Shadow	False

This icon is a line icon which means that while it appears in the palette as an icon, it behaves somewhat like a line on the window. By clicking on the ends of the bus and dragging, a bus icon can be resized and reshaped. Figure 4.56 shows several bus line icons, demonstrating the versatility of the window representation of a bus.



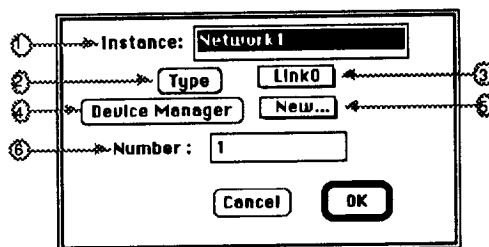
The Bus "Line Icon"
Figure 4.56

The methods for the bus icon are:

Method Name	Function
Deleted	Destroys the device data structure associated with the icon.
Duplicated	Attaches a device data structure to the new duplicate icon which is a copy of the one associated with the icon being duplicated. However, the name will be changed so that it is unique from all other names in the window.
Pasted	Ensures that the name of the device data structure associated with the icon being pasted is distinct from all other devices in the

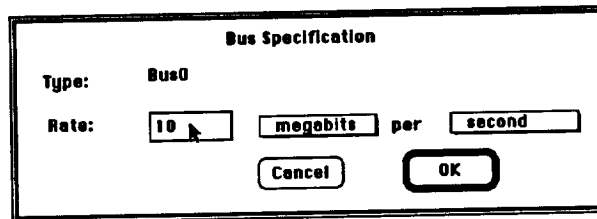
	destination window. The name will be changed if necessary.
Gender	Ensures that the name of the icon's device data structure is on the window's routing tables if there are any connectors to the icon.
Save	Saves the device data structure associated with the icon to the model file.
Restore	Reads a device data structure definition from the model file and recreates the structure. This structure will then be associated with the proper window and icon.
Connected	Allows connectors to be drawn to and from the icon.
DoubleClick	Double clicking this icon will cause the device data structure to be created for this icon if it does not yet exist. Additionally, the device dialog will be displayed for the structure (see below).
GetInfo	Selecting this icon and choosing the "Get Info" item of the Edit menu will cause the device data structure to be created for this icon if it does not yet exist. Additionally, the device type dialog will be displayed for the structure (see below).

As mentioned in the method list, double clicking on a bus will display the bus dialog shown in Figure 4.57. This dialog contains the information which defines the bus. The name (field 1), in this case "Network1", is the name the icon will be given. This name must be distinct from all other device names in the same window. The bus type popup menu (field 3) will contain a list of all defined bus types with the type of this bus being shown. A bus type defines the transmission characteristics of the bus. Clicking on the "Type" button (field 2) will display the bus type dialog explained below. The popup menu next to the "Device Manager" button (field 5) will contain a list of all defined device managers with the manager of this bus showing. Clicking on the "Device Manager" button (field 4) will display the device manager dialog. Managers are explained in Section 4.6. Finally, the number field (field 6) shows the number of channels on this bus.



The Bus Dialog
Figure 4.57

The bus type dialog shown in Figure 4.58 is accessed by clicking on the "Type" button of the bus pool dialog or by selecting a bus icon and choosing the "Get Info" item in the Edit menu. A bus type determines the transmission rate of the bus. A given type may be used by any number of bus pools. That is, each bus icon represents a unique named bus pool, but these pools may be of the same type. The bus type name must be distinct from all other bus types. The transmission rate of the bus type is expressed as an amount, a quantity unit and a time unit. Both unit fields are popup menus.



The Bus Type Dialog
Figure 4.58

4.5.3. THE NODE ICON

The node icon on the Hardware window allows the user to attach another hardware window to an existing hardware window. This icon allows the hardware architecture of the model to be defined as a hierarchy of hardware windows. This icon has its object fields set to:

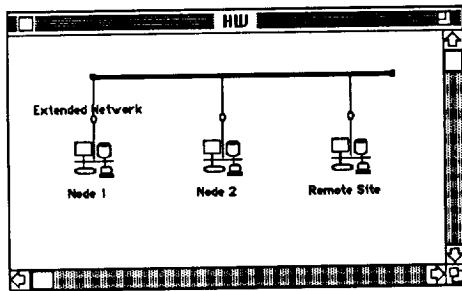
Field Name	Setting
Genus	Node
Species	Node
Type	BlockIcon
Shadow	False

The methods for the node icon are:

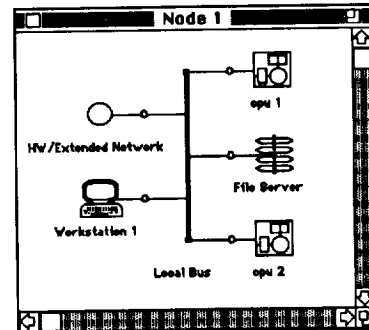
Method Name	Function
Deleted	Destroys the hardware window associated with the node and all its contents.
Duplicated	Currently not allowed.
Pasted	Currently not allowed.
Gender	No function
Save	Saves the information necessary to link this icon with its window in the model file.
Restore	Reads the information saved (see above) and reattaches this icon to its window. The model save/restore mechanism will ensure that the window being attached has already been restored.
Connected	Allows connectors to be drawn to and from the icon.
DoubleClick	Double clicking this icon will cause a new hardware window with a unique name to be created and attached to this icon.
GetInfo	Selecting this icon and choosing the "Get Info" item of the Edit menu will cause a dialog to be displayed which allows the associated window name to be changed. The new name must be distinct from all other window names in the model.

Figure 4.59 shows a hardware window which contains a bus and three node icons. This diagram represents a simple network with two local (and similar) nodes, and a remote node. Figures 4.60 and 4.61 show the hardware windows associated with the "Node1" and "Node2" icons in the "HW" window. Note that icon names must be unique within a window, but not across windows.

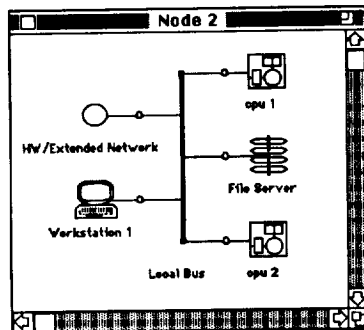
This is demonstrated by the contents of "Node1" and "Node2" which do not differ internally. Figure 4.62 shows the window associated with the "Remote Site" icon in the "HW" window and Figure 4.63 shows the "Local Server" window which is nested in the "Remote Site". This series of windows demonstrates the hierarchical nesting of hardware windows. Figure 4.64 gives a hierarchical view of the relationships between these windows. Note that there is no limit on the number of hardware windows or the depth of nesting.



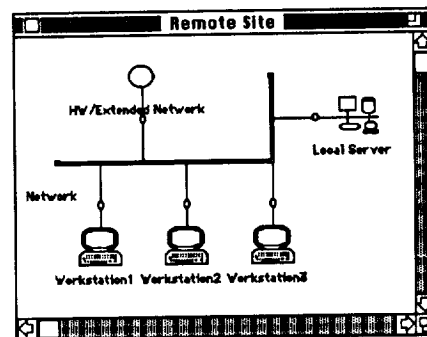
Hardware Window With Nodes
Figure 4.59



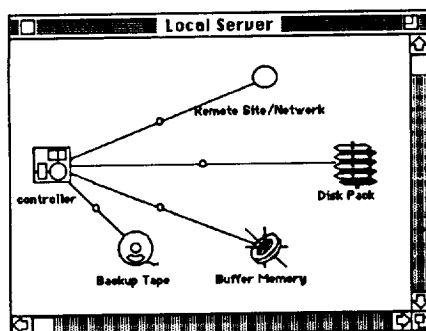
A Nested Hardware Window
Figure 4.60



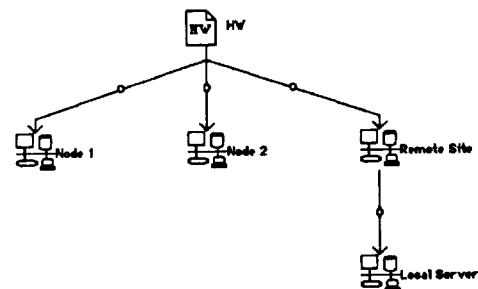
A Nested Hardware Window
Figure 4.61



A Nested Hardware Window
Figure 4.62



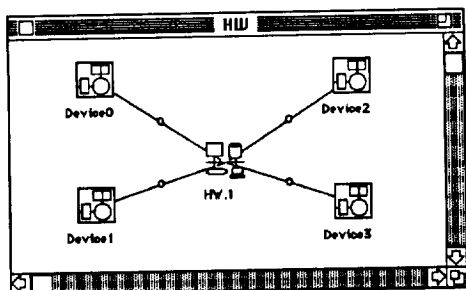
A Nested Hardware Window
Figure 4.63



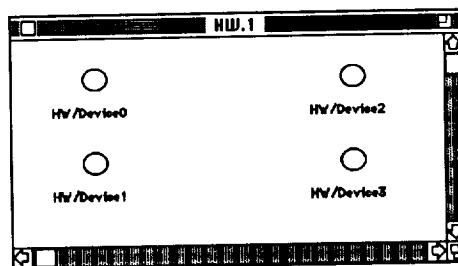
A Sample Hardware Window
Hierarchy
Figure 4.64

4.5.4. THE BOUNDARY ICON

The boundary icon does not appear on the hardware window palette. This icon appears on all hardware windows except the topmost in the hardware hierarchy. One boundary icon is added to each nested window for each connector made to the node icon associated with this window. Since there may be several boundary icons in a window, each of which provides a mechanism to connect to a different device in the parent window (the one with the node icon), the names of the boundary icons must correspond to the icon from which the boundary connects (by way of the node icon). Figure 4.65 shows a hardware window with one node icon with many icons connected to it. Figure 4.66 shows the window associated with the node icon. By examining these two Figures it should be obvious that the name of the boundary created by making a connector to a node icon denotes the icon on the other end of the connector. Part of the name also denotes the window in which this other icon resides.



A Hardware Window
Figure4.65



A Hardware Window With Multiple
Boundary Icons
Figure 4.66

The boundary icon has its object fields set to:

Field Name	Setting
Genus	NodeBoundary
Species	NodeBoundary
Type	BlockIcon
Shadow	False

The methods for the boundary icon are:

Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	No function.

Save	Saves the required information to allow the icon and all connectors to it to be restored to the model file.
Restore	Recreates the boundary icon and all connectors to it.
Connected	Allows connectors to be drawn to and from the icon.
DoubleClick	Not allowed.
GetInfo	Not allowed.

The boundary icon's only function is to provide a way to extend connectors drawn to a node icon into the node's associated window. Thus, this icon is created by Pedestal™ automatically, and the user is only allowed to reposition it and draw connectors to/from it.

4.5.5. HARDWARE CONNECTORS

Hardware connectors always appear as undirected lines, thus the object field settings for connectors are:

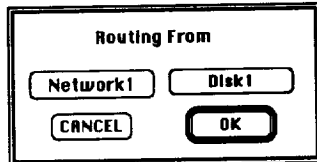
Field Name	Setting
Flag1Icon	Not used.
Flag2Icon	Not used.
HotSpotIcon	Not used.

The method routines for a connector are:

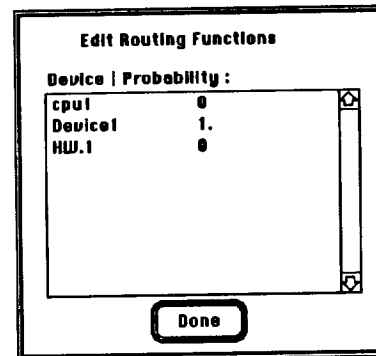
Method Name	Function
Deleted	Destroys the routing list associated with the connector.
Duplicated	Attaches a routing list to the new duplicate connector. Since the names of the icons it connects may be changed (due to conflict with existing devices) the routing table may not be an exact copy of the one in the connector being duplicated.
Pasted	Creates and updates the routing list associated with the connector so that it contains the proper names.
Gender	Not used.
Save	Saves the routing list associated with the connector to the model file.
Restore	Reads a routing list definition from the model file and recreates the structure. This structure will then be associated with the proper connector.
DoubleClick	Double clicking a connector will cause a routing list to be created for this connector if it does not yet exist. Additionally, the first routing dialog will be displayed (see below).

As mentioned in the method list, double clicking on a connector will display the first routing dialog. The dialog shown in Figure 4.67 corresponds to the connector between the disk icon and the bus on the hardware window shown previously in Figure 4.47. Since connectors are bidirectional and routing may be different based on which way the message is being sent, there are two buttons on the dialog. Each button corresponds to a different routing list. The two buttons labeled "Network1" and "Disk1", if clicked, will show the routing list dialog for messages being sent from the bus and disk respectively. Note that the names on the buttons on this dialog will be different for every connector.

If we click on the button labeled "Network1" in the first routing dialog, we will be shown the routing list dialog presented in Figure 4.68. This dialog lists all the valid destinations for a message currently on the "Network1" bus, along with the probability that the communication path associated with this connector will be taken for messages being sent to that destination. For example, the routing list shown in the dialog says that all messages on the "Network1" bus with a destination device of "Device1" will take this path with probability 1, while messages on "Network1" heading for "cpu1" or "HW.1" (a window) will take this path with probability 0.



The First Routing Dialog
Figure 4.67

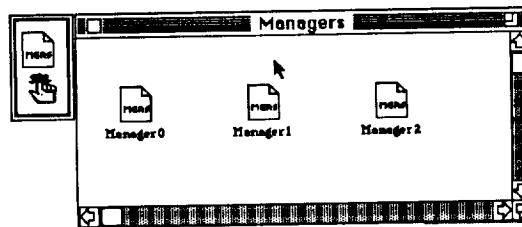


The Routing List Dialog
Figure 4.68

By default, the probabilities in the routing lists will be one for devices on the connector and zero for all others. To change a probability in a routing list, double click on the probability you wish to change. A simple dialog will be displayed which will allow the value to be changed. Note that expressions (Chapter 3) may be used to implement more complex routing.

4.6. MANAGER WINDOW

The manager window shown in Figure 4.69 has an icon for every manager which is currently defined and provides a convenient method for creating or modifying managers. New managers are created here by selecting the manager icon from the palette and placing it in the window, just as icons are placed in any other window that has a palette. When a manager is created from another window, an icon is automatically placed here for that manager. The note icon allows the user to append a textual description; refer to Section 4.1.1.5 for a complete description of the note icon.



The Manager Window
Figure 4.69

The object fields for the manager window are set as follows:

Field Name	Setting
WindowType	Task
ConnectorType	NoConnector
CanEdit	True
CanGetInfo	True
CanMouse	True
CanReduce	True
CanTile	True

The connector type denotes that connectors are not used on the manager window. The manager window methods are:

Method Name	Function
ConnectorDoubleClick	Not allowed.
PlaceConnector	Not allowed.
Activate	No function.

4.6.1. THE MANAGER ICON

The manager icon allows the user to create and/or modify tasks. This icon has its object fields set to:

Field Name	Setting
Genus	Manager
Species	Manager
Type	BlockIcon
Shadow	False {Variable}

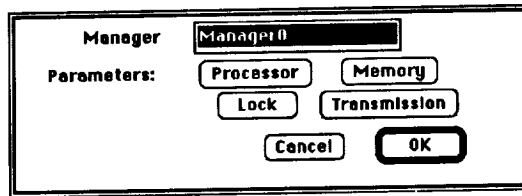
The shadow field is not used in the manager window. The methods for the manager icon are:

Method Name	Function
Deleted	Delete the manager associated with the icon and remove all references to this manager by hardware devices.
Duplicated	Create a new manager and associate it with the new duplicate icon. The characteristics of the new manager will be the same as the manager being duplicated, only the name will change. The manager names must be distinct.
Pasted	Places a previously duplicated or cut manager icon on the window.
Gender	No function.
Save	Saves the manager information associated with the icon to the model file.
Restore	Reads a manager definition from the model file and associates it with the proper window and icon.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the manager dialog to be displayed (see below).
GetInfo	Same as double click.

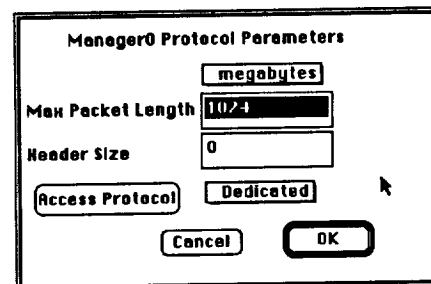
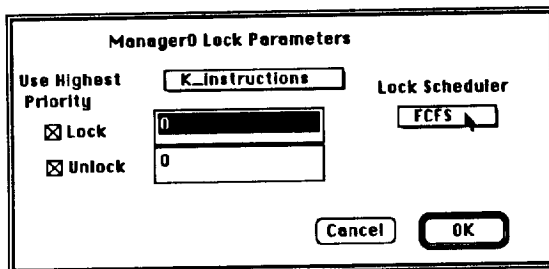
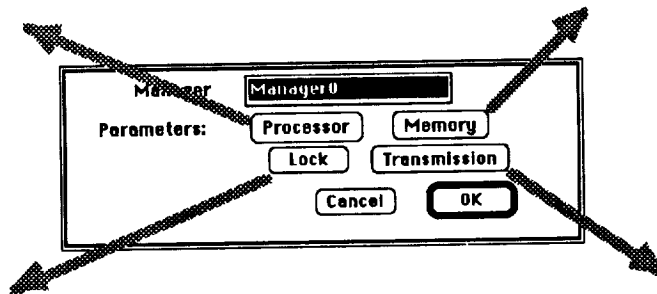
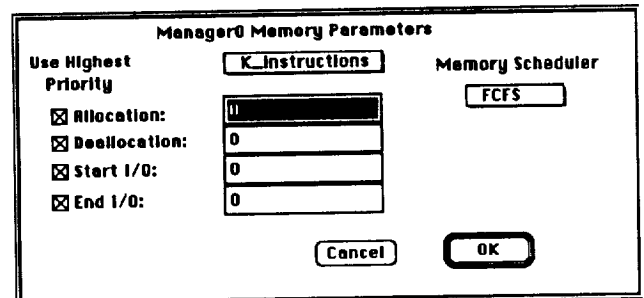
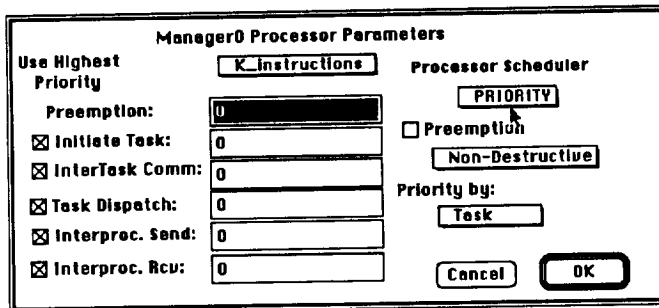
As mentioned in the method list, double clicking on a manager icon will display the manager dialog shown in Figure 4.70. This dialog may also be reached by clicking on the "Manager" button on the device dialogs (see Section 4.5.2).

The first field on the manager dialog allows the name of the manager to be changed. All devices which reference the manager will still reference it after the name is changed. The four buttons on the dialog box provide access to the

attributes of the manager specific to the type of device named on the button. Figure 4.71 below shows the four dialogs. Each dialog except for Lock Parameters may be accessed through the individual hardware devices (on the Hardware Window) as well as from the manager window; the lock overheads may be accessed from the lock request/relinquish dialogs (see Sections 4.1.2.8 and 4.1.2.9).



The Manager Dialog
Figure 4.70

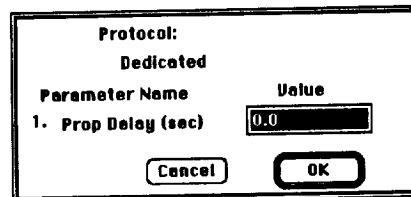


Manager Parameters
Figure 4.71

For the processor, memory, and lock parameters a popup menu appears at the top of the editable text field to specify the units in which the overheads are expressed.

Each of these three types of device has a scheduler associated with its management which has a default of First Come First Served (FCFS). In Figure 4.71 the processor manager dialog shows the additional fields that are display when priority scheduling is chosen. Preemption is chosen by checking the checkbox and a popup allows further declaration of destructive or non-destructive preemption. The bottom popup allows selection of priority by task, workload, or module. The checkboxes that appear to the left of the parameter names indicates whether the execution of overhead takes precedence over other jobs. By default these overheads are handled at the highest priority.

An access protocol is associated with the transmission parameters; all available protocols appear in a popup menu with the selected protocol visible. Currently, the only protocol implemented is Dedicated. By clicking on the Access Protocol button, the user may view and edit the parameters of the selected protocol function. Figure 4.72 is the dialog in which the Dedicated protocol may be edited.



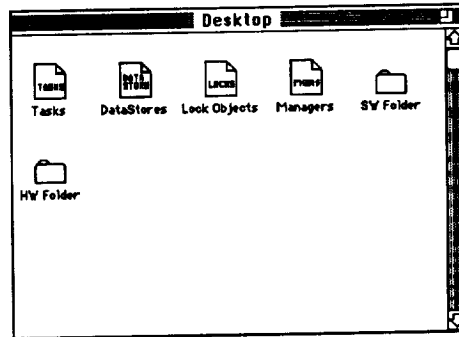
The image shows a dialog box titled "Protocol:". Inside, it says "Dedicated". Below this is a table with two columns: "Parameter Name" and "Value". The first row in the table is "1. Prop Delay (sec)" with a value of "0.0" in a text box. At the bottom of the dialog are two buttons: "Cancel" and "OK".

Parameter Name	Value
1. Prop Delay (sec)	0.0

Protocol Dialog
Figure 4.72

4.7. THE DESKTOP WINDOW

The desktop window, shown in Figure 4.73, is supplied by Pedestal™ for navigational purposes and is similar in concept to the desktop on the Macintosh. By double clicking on the icons in the desktop, the user may view any window in the currently loaded model.



The Desktop Window
Figure 4.73

Unlike most window types in Pedestal™, there is only one Desktop window and the permitted actions are few. Also unlike other Pedestal™ windows, the Desktop window has no palette. The icons on this window are produced by Pedestal™ and the user may not add or remove any of them. Additionally, there are no connectors on the Desktop window.

The object fields for the desktop window are set as follows:

Field Name	Setting
WindowType	Desktop
ConnectorType	none
CanEdit	False
CanGetInfo	False
CanMouse	True
CanReduce	False
CanTile	False

These settings only allow the user to double click, select and reposition icons. The methods for the Desktop window are:

Method Name	Function
ConnectorDoubleClick	none
PlaceConnector	none
Activate	none

None of the window method routines have a function since there are no connectors and the window requires no special activation.

Since there are no connectors on the Desktop window there is no need to examine the connector object fields or methods.



Tasks

4.7.1. THE TASK WINDOW ICON

The task window icon on the Desktop window allows the user to display the Task window (see Section 4.2). Since the user is only allowed to double click or move icons on the Desktop window, the field settings for this icon can not change. This icon has its object fields set to:

Field Name	Setting
Genus	TaskWindow
Species	TaskWindow
Type	BlockIcon
Shadow	False

The methods for the task window icon are:

Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	Not allowed.
Save	Not allowed. The Desktop window and its icons are created as a by-product of the creation of the other windows in a model. Therefore it does not need a save or restore method since the act of restoring the rest of the model will automatically restore the Desktop window and its icons.
Restore	Not allowed. See above.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the Task window (see Section 4.2) to be displayed and selected.
GetInfo	Not allowed.



4.7.2. DataStores THE DATASTORE WINDOW ICON

The datastore window icon on the Desktop window allows the user to display the Datastore window (see Section 4.4). Since the user is only allowed to double click or move icons on the Desktop window, the field settings for this icon can not change. This icon has its object fields set to:

Field Name	Setting
Genus	DatastoreWindow
Species	DatastoreWindow
Type	BlockIcon
Shadow	False

The methods for the datastore window icon are:

Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	Not allowed.
Save	Not allowed. The Desktop window and its icons are created as a by-product of the creation of the other windows in a model. Therefore it does not need a save or restore method since the act of restoring the rest of the model will automatically restore the Desktop window and its icons.
Restore	Not allowed. See above.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the Datastore window (see Section 4.4) to be displayed and selected.
GetInfo	Not allowed.



Lock Objects

4.7.3. THE LOCK WINDOW ICON

The lock window icon on the Desktop window allows the user to display the Lock window (see Section 4.3). Since the user is only allowed to double click or move icons on the Desktop window, the field settings for this icon can not change. This icon has its object fields set to:

Field Name	Setting
Genus	LockWindow
Species	LockWindow
Type	BlockIcon
Shadow	False

The methods for the lock window icon are:

Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	Not allowed.
Save	Not allowed. The Desktop window and its icons are created as a by-product of the creation of the other windows in a model. Therefore it does not need a save or restore method since the act of restoring the rest of the model will automatically restore the Desktop window and its icons.
Restore	Not allowed. See above.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the Lock window (see Section 4.3) to be displayed and selected.
GetInfo	Not allowed.



4.7.4. Managers THE MANAGER WINDOW ICON

The manager window icon on the Desktop window allows the user to display the Manager window (see Section 4.6). Since the user is only allowed to double click or move icons on the Desktop window, the field settings for this icon can not change. This icon has its object fields set to:

Field Name	Setting
Genus	ManagerWindow
Species	ManagerWindow
Type	BlockIcon
Shadow	False

The methods for the manager window icon are:

Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	Not allowed.
Save	Not allowed. The Desktop window and its icons are created as a by-product of the creation of the other windows in a model. Therefore it does not need a save or restore method since the act of restoring the rest of the model will automatically restore the Desktop window and its icons.
Restore	Not allowed. See above.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the Manager window (see Section 4.6) to be displayed and selected.
GetInfo	Not allowed.



SW Folder

4.7.5.

THE SOFTWARE SIDEVIEW WINDOW ICON

The software sideview window icon on the Desktop window allows the user to display the Software Sideview window (see Section 4.8). Since the user is only allowed to double click or move icons on the Desktop window, the field settings for this icon can not change. This icon has its object fields set to:

Field Name	Setting
Genus	SWSideviewWindow
Species	SWSideviewWindow
Type	BlockIcon
Shadow	False

The methods for the software sideview window icon are:

Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	Not allowed.
Save	Not allowed. The Desktop window and its icons are created as a by-product of the creation of the other windows in a model. Therefore it does not need a save or restore method since the act of restoring the rest of the model will automatically restore the Desktop window and its icons.
Restore	Not allowed. See above.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the Software Sideview window (see Section 4.8) to be displayed and selected.
GetInfo	Not allowed.



HW Folder

4.7.6. THE HARDWARE SIDEVIEW WINDOW ICON

The hardware sideview window icon on the Desktop window allows the user to display the Hardware Sideview window (see Section 4.8). Since the user is only allowed to double click or move icons on the Desktop window, the field settings for this icon can not change. This icon has its object fields set to:

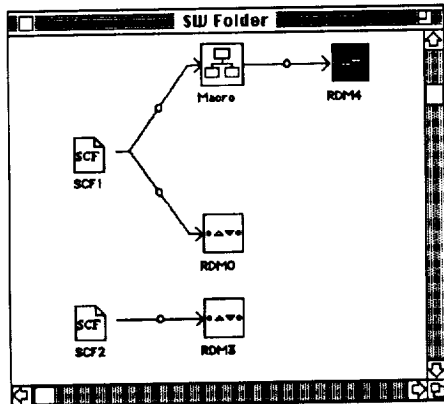
Field Name	Setting
Genus	HWSideviewWindow
Species	HWSideviewWindow
Type	BlockIcon
Shadow	False

The methods for the hardware sideview window icon are:

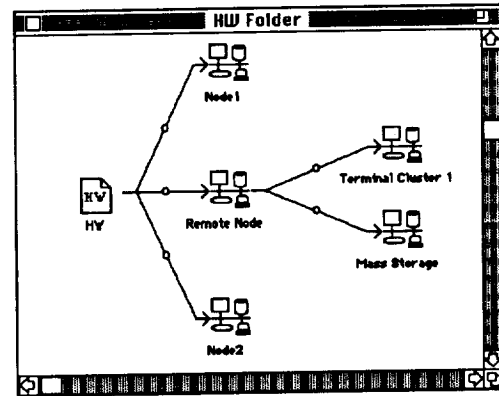
Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	Not allowed.
Save	Not allowed. The Desktop window and its icons are created as a by-product of the creation of the other windows in a model. Therefore it does not need a save or restore method since the act of restoring the rest of the model will automatically restore the Desktop window and its icons.
Restore	Not allowed. See above.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the Hardware Sideview window (see Section 4.8) to be displayed and selected.
GetInfo	Not allowed.

4.8. SIDEVIEW WINDOWS

The hardware and software folder icons on the desktop window (see Section 4.7) display sideview windows when double clicked. These sideview windows show the hierarchical arrangement of the software and hardware windows respectively (see Sections 4.1.1 and 4.1.2). Figures 4.74 and 4.75 show sample software and hardware sideviews. The icons in these windows correspond to actual windows. If double clicked, Pedestal™ will display the appropriate window. No operations other than double clicking icons and scrolling are allowed on sideview windows.



A Software Sideview Window
Figure 4.74



A Hardware Sideview Window
Figure 4.75

The object fields for the sideview windows are set as follows:

Field Name	Setting
WindowType	Sideview
ConnectorType	HorizontalArrow
CanEdit	False
CanGetInfo	False
CanMouse	True
CanReduce	True
CanTile	True

The methods for sideview windows are:

Method Name	Function
ConnectorDoubleClick	Not allowed.
PlaceConnector	Not allowed.
Activate	No function.

The object fields for all the icons on a sideview window are:

Field Name	Setting
Genus	SideviewIcon
Species	SideviewIcon
Type	BlockIcon
Shadow	False

The methods for the icons on a sideview window are:

Method Name	Function
Deleted	Not allowed.
Duplicated	Not allowed.
Pasted	Not allowed.
Gender	Not allowed.
Save	Not allowed. A sideview window and its icons are created as a by-product of the creation of the other windows in a model. Therefore it does not need a save or restore method since the act of restoring the rest of the model will automatically restore the sideview windows and their icons.
Restore	Not allowed. See above.
Connected	Not allowed.
DoubleClick	Double clicking this icon will cause the the associated window to be displayed and selected.
GetInfo	Not allowed.

5. ASSOCIATION LISTS

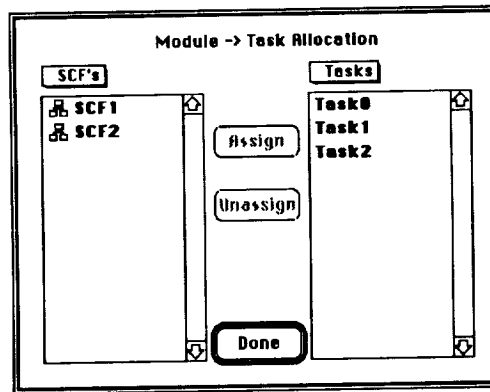
One of the major features of Pedestal™ is the ability to define the software and hardware parts of a model independently. This keeps changes in hardware from requiring changes in the software. However, to run a simulation of the model the software and hardware must be related. On which memory device does the "named data store" datastore reside? What processor do execution requests in the modules of "task0" use? The association list (or mapping) utility provides an easy way to make these assignments.

The association list utility is also used to allow a simple way to modify the module to task and device to manager mappings which deal solely with software and hardware concepts respectively. These settings may be changed at the individual icons, but for large models it may be cumbersome to page through the diagrams to find the right icons to double click, especially if a large number of these assignments are to be changed. Using the association list to make these kinds of changes is usually simpler.

The mechanics of the association list dialog and the particular function with respect to the different mappings is discussed in the following Sections.

5.1. ASSOCIATION LIST DIALOG

The association list dialog is used to perform mappings. Figure 5.1 shows the association list dialog for the module to task mapping. Note that the dialog works the same for all mappings, only the names of the fields change. At the top of the dialog is a name which identifies the specific association being made; in this case "Module -> Task Allocation". The list and popup menu on the left are concerned with the things being assigned; we will call these objects. The ones on the right concern the things being assigned to; we will call these classes. A mapping is made by assigning all objects to classes.



The Module to Task Mapping Dialog
Figure 5.1

The set of objects for a given mapping may be hierarchical. In the mapping of Figure 5.1, the objects are hierarchical. The topmost level of the hierarchy consists of the names of all defined SCF windows. The next level contains the names of all defined modules. Double clicking on one of the SCF names will cause that window's name to be added to the object popup menu as the selected item and the object list will then display the names of all modules within that window. Since the modules are the bottom of this particular hierarchy, no icon is displayed next to their names and they can not be double clicked. To move back up the hierarchy, select "SCF's" in the object popup menu and the names of the SCF windows will replace the names of the modules of the previously selected SCF window. The previously selected window's name will be removed from the popup menu.

Note that the dialog places no limit on the number of levels of the hierarchy and the icon to the left of an object name is used to denote that object has children. Every time you double click on an object and step further down the hierarchy, the item clicked on is added to the object popup menu. This allows you to step out to any higher level in one step by choosing the name of the object at the level you want to step out to.

Objects in the object list will be shown in italics if they are currently assigned to a class. If a nonleaf object (one with an icon to the left) is in italics, then all its children objects (and their children) are currently assigned.

The class list may also be hierarchical. The class list also uses an icon to the left of a name to indicate that a class has children. Assignments may only be made to simple classes (no children). Navigation through the class list works in the same manner as that for the object list with one exception. A simple class may be double clicked to display the names of the objects currently assigned to that class. Note that only the names of objects without children appear, since they are the only objects which can actually be mapped (higher level objects are really only classifiers not objects).

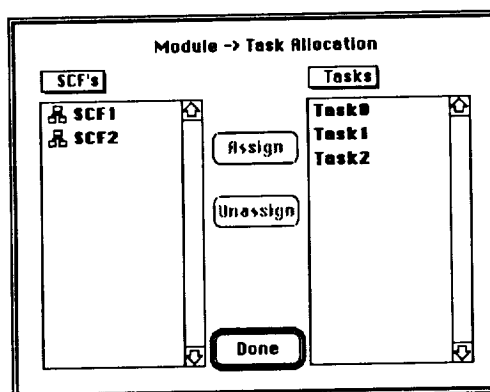
To perform an assignment, click on an object in the object list and on a simple class in the class list to select the object and the class to which it is to be assigned. Next, click on the "Assign" button. This will cause the specific object to be assigned to the selected class. If the object has children, they will all be assigned to the specified class. If the object, or any of its children, were previously assigned, they will be reassigned to the specified class.

To remove an assignment, select the object to be unassigned and click on the "Unassign" button. If the object selected has children, all of the children will also be unassigned.

5.2. MODULE -> TASK

All modules must be grouped into tasks before a model can be simulated. This grouping is done via the module to task mapping. The module to task mapping can be performed directly from the module dialog (Section 4.1.1.3) or from the "Module -> Task" item of the map menu (Section 2.5). Refer to Sections 4.1.1.3 and 4.2 for discussions of modules and tasks.

Selecting the "Module -> Task" item of the map menu will display the module to task mapping dialog shown in Figure 5.2. Initially, the left hand list will display the names of all SCFs defined in the current model and the right hand list will display the names of all defined tasks. If all of the modules in a window are currently mapped to tasks, the window name will appear in italics.



The Module to Task Mapping Dialog
Figure 5.2

The mapping may be made at this level if desired. To do so, select one or more window names in the left hand list and select one task in the right hand list and then click on the "Assign" button. This will cause all modules in the selected windows to be assigned to the selected task. Any modules which were previously assigned to a different task will be reassigned to the current task.

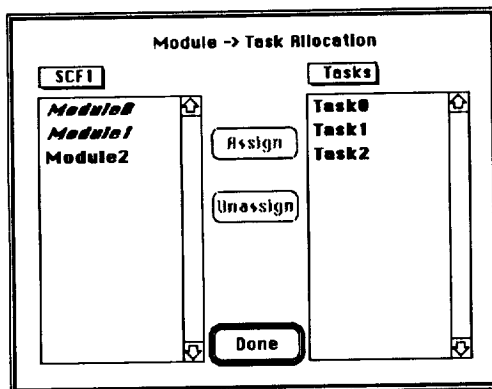
Removing mappings may also be done at the window level. Select one or more windows in the window list and then click on the "Unassign" button. All modules in the selected windows will no longer be mapped to any task.

This mapping may also be done on a module by module basis. Double clicking on a window name in the left hand list will cause that window's name to be added to the left hand popup menu and the left hand list will now display the names of all modules in that SCF. Modules which are already assigned to a task will be shown in italics. Select the modules in the left hand list you wish to map to a common task, select the task in the right hand list, and click the "Assign" button to map at this level. All selected modules will now be assigned to the selected task even if they were previously assigned to a different task. Removing an assignment is done by selecting the modules in the left hand list

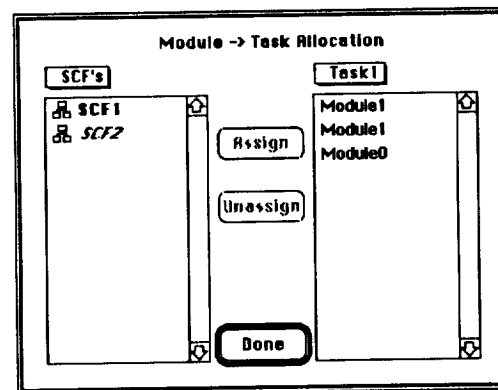
which are to be unassigned and clicking the "Unassign" button. Figure 5.3 shows the module to task allocation dialog from Figure 5.2 after "SCF1" was double clicked. Note that "Module0" and "Module1" are currently mapped to tasks.

To map the modules from another SCF you must select the "SCF's" item in the left hand popup menu. Then the left hand list will once again show the window names. Now you can double click on a different SCF to map its modules individually.

At any time, double clicking on a task in the right hand list will cause the name of the task clicked on to be added to the right hand popup menu. Additionally, the right hand list will now display the names of all modules currently mapped to the selected task. By selecting the "Tasks" item in the right hand popup menu you can get back to the list of tasks.



Viewing the Modules in an SCF
Figure 5.3

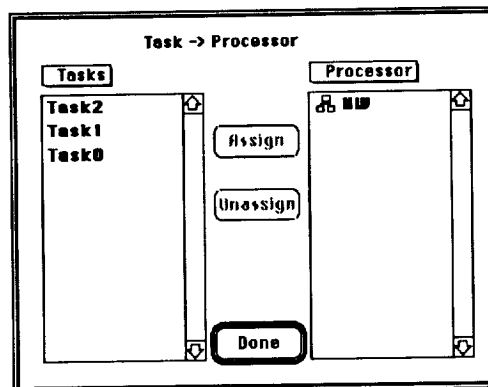


Viewing the Modules Assigned to a Task
Figure 5.4

5.3. TASK -> PROCESSOR

All tasks must be assigned to a processor. Defining which tasks use which processors is done via the task to processor mapping. The task to processor mapping can only be performed from the "Task -> Processor" item of the map menu (Section 2.5). Refer to Sections 4.2 and 4.5.2.1 for discussions of tasks and processors. Note that more than one task may be assigned to one processor.

Selecting the "Task -> Processor" item of the map menu will display the task to processor mapping dialog shown in Figure 5.5. Initially, the left hand list will display the names of all currently defined tasks, and the right hand list will display the name of the top level hardware window "HW". Tasks which are already assigned to a processor will be shown in italics.

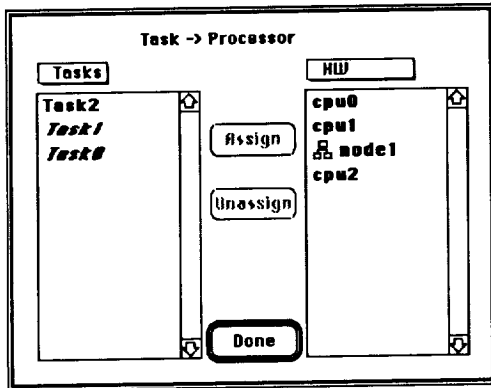


The Task to Processor Mapping Dialog
Figure 5.5

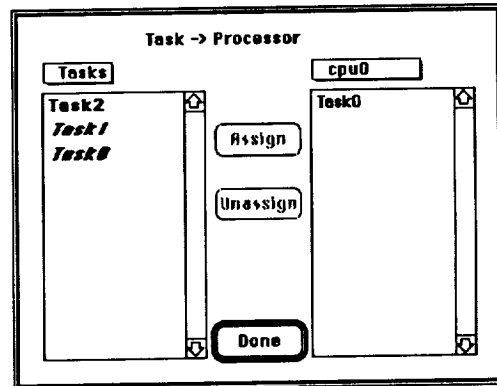
The right hand list is hierarchical and reflects the hardware hierarchy. Double clicking on "HW" will cause "HW" to be added to the right hand popup menu and the names of all the processors and nodes in the "HW" window will appear in the list. Nodes in the right hand window are denoted by a small icon to the left of the name. Double clicking on a node in the right hand list will have a similar effect, displaying the processors and nodes within that node. Double clicking on a processor's name in the right hand list will cause the names of all tasks currently assigned to that processor to be displayed. To move back to a higher level in the hierarchy, use the right hand popup menu. The menu will list all the nodes in the path down to the currently displayed node or processor. Choosing a name will cause the chosen node's contents to be displayed in the right hand list. Figure 5.6 shows the contents of the right hand list after "HW" is double clicked. Figure 5.7 shows the contents of the right hand list after one of the processors in the "HW" window is double clicked.

Assignment is done by selecting one or more tasks in the left hand list, selecting a processor in the right hand list, and clicking on the "Assign" button. Note that the "Assign" button will be dimmed (disabled) if a node is selected in

the right hand list. All tasks which were selected will be assigned to the selected processor even if some of the tasks were previously assigned. Selecting one or more tasks and clicking the "Unassign" button will cause the selected tasks to be disassociated from any processor.



Viewing the Contents of "HW"
Figure 5.6

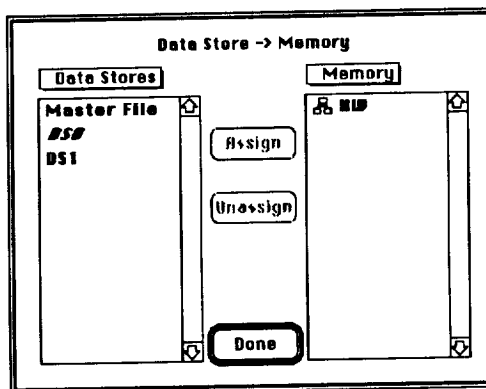


Viewing the Tasks Assigned to a
Processor
Figure 5.7

5.4. DATASTORE -> MEMORY

All datastores must be assigned to a memory device. Defining which datastores use which memories is done via the datastore to memory mapping. The datastore to memory mapping can only be performed from the "Datastore -> Memory" item of the map menu (Section 2.5). Refer to Sections 4.4 and 4.5.2.2 for discussions of datastores and memories. Note that more than one datastore may be assigned to one memory.

Selecting the "Datastore -> Memory" item of the map menu will display the datastore to memory mapping dialog shown in Figure 5.8. Initially, the left hand list will display the names of all currently defined datastores, and the right hand list will display the name of the top level hardware window "HW". Datastores which are already assigned to a memory will be shown in italics.

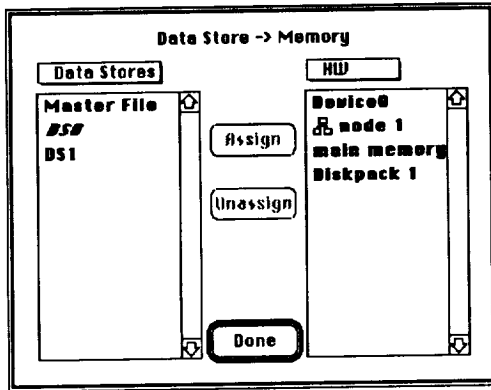


The Datastore to Memory Mapping Dialog
Figure 5.8

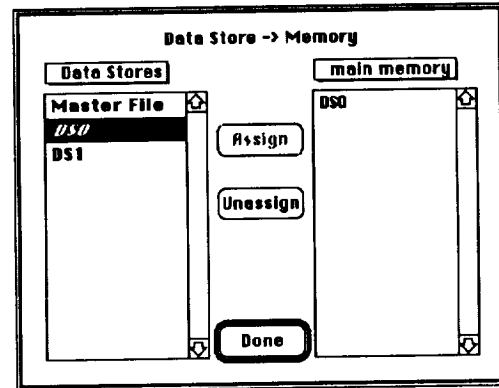
The right hand list is hierarchical and reflects the hardware hierarchy. Double clicking on "HW" will cause "HW" to be added to the right hand popup menu and the names of all the memories and nodes in the "HW" window will appear in the list. Nodes in the right hand window are denoted by a small icon to the left of the name. Double clicking on a node in the right hand list will have a similar effect, displaying the memories and nodes within that node. Double clicking on a memory's name in the right hand list will cause the names of all datastores currently assigned to that memory to be displayed. To move back to a higher level in the hierarchy, use the right hand popup menu. The menu will list all the nodes in the path down to the currently displayed node or memory. Choosing a name will cause the chosen node's contents to be displayed in the right hand list. Figure 5.9 shows the contents of the right hand list after "HW" is double clicked. Figure 5.10 shows the contents of the right hand list after one of the memories in the "HW" window is double clicked.

Assignment is done by selecting one or more datastores in the left hand list, selecting a memory in the right hand list, and clicking on the "Assign" button. Note that the "Assign" button will be dimmed (disabled) if a node is selected in

the right hand list. All datastores which were selected will be assigned to the selected memory even if some of the datastores were previously assigned. Selecting one or more datastores and clicking the "Unassign" button will cause the selected datastores to be disassociated from any memory.



Viewing the Contents of "HW"
Figure 5.9

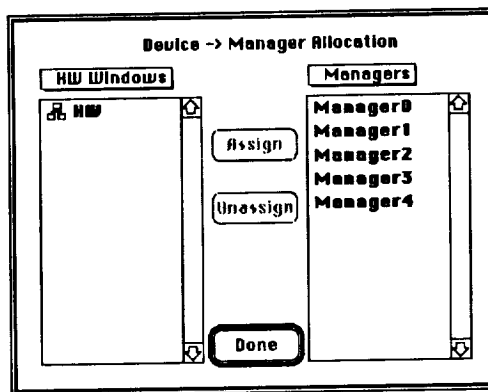


Viewing the Datastores Assigned to
a Memory
Figure 5.10

5.5. DEVICE -> MANAGER

Devices may use and share managers. Defining which devices use which managers is done via the device to manager mapping. The device to manager mapping can be performed directly from the device dialog (Section 4.5.2) or from the "Device -> Manager" item of the map menu (Section 2.5). Refer to Section 4.5.2 and Chapter 5 for discussions of devices and managers. Note that devices need not be assigned to any manager.

Selecting the "Device -> Manager" item of the map menu will display the device to manager mapping dialog shown in Figure 5.11. Initially, the left hand list will display the names of the top level hardware window, "HW", of the current model and the right hand list will display the names of all defined managers. If all devices are currently mapped to managers, the window name will appear in italics.



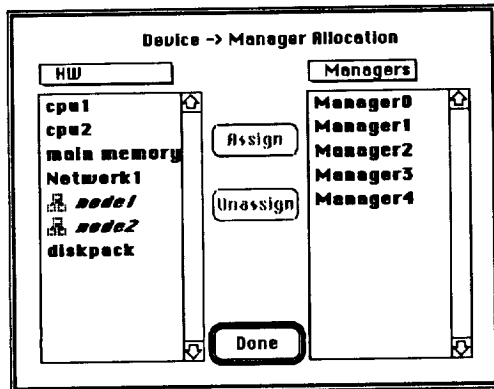
The Device to Manager Mapping Dialog
Figure 5.11

If you wish to map all devices to one manager, select "HW", select the desired manager, and click on the "Assign" button. All devices will now be mapped to the selected manager, even if they were previously mapped to a different manager. If you select "HW" and click on the "Unassign" button, all devices will be dissassociated from managers.

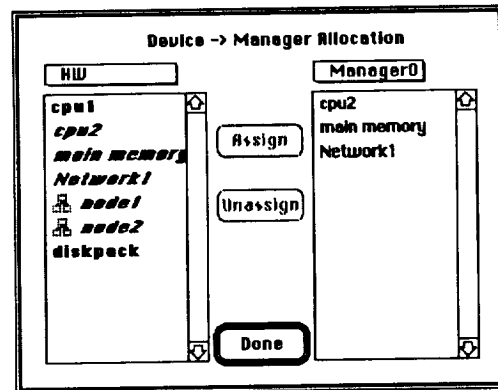
The left hand list is hierarchical and reflects the hardware hierarchy. Double clicking on "HW" will cause "HW" to be added to the left hand popup menu and the names of all the devices and nodes in the "HW" window will appear in the list. All devices which are currently assigned to a manager will appear in italics. If all devices in a node are assigned, the node will appear in italics (note: all devices in nodes within the node must also be assigned.). Assignment may be done at this level by selecting node and/or device names and a manager and clicking on the "Assign" button. Unassignment is done by selecting the proper devices and/or nodes and clicking on the "Unassign" button. Nodes in the left hand window are denoted by a small icon to the left of the name. Any node may be double clicked to have its contents displayed. To

move back to a higher level in the hierarchy, use the left hand side popup menu. The menu will list all the nodes in the path down to the currently displayed node. Choosing a name will cause the chosen nodes contents to be displayed in the left hand list. Figure 5.12 shows the contents of the left hand list after "HW" is double clicked.

At any time, double clicking on a manager in the right hand list will cause the name of that manager to be added to the right hand popup menu. Additionally, the right hand list will now display the names of all devices currently mapped to the selected manager. By selecting the "Managers" item in the right hand popup menu you can get back to the list of managers.



Viewing the Contents of "HW"
Figure 5.12



Viewing the Devices Assigned to a
Manager
Figure 5.13

6. STATISTICS

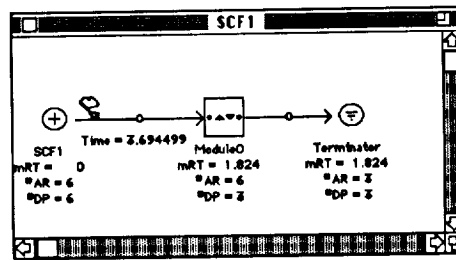
Of what use is a simulator if it does not produce statistics? Pedestal™ collects and calculates a number of statistics for all parts of a model and allows the user to define new statistics to be collected. To reduce the run time overhead of calculating statistics, the user may choose the specific statistics to be calculated during the stimulation of the model.

The major categories of statistics available from Pedestal™ are response , service , and queue times. All categories are not applicable in all situations; e.g., utilization of locks (as they implemented in Pedestal™) has no meaning. There are exclusive and non exclusive requests for a lock (see Sections 4.1.2.8 and 4.1.2.9) which is, in effect, a device with dynamically changing capacity. Mean response times, which represent the sum of the mean queue and mean service times, are reported for hardware devices but max and min response times are not available. This is because queue and service statistics are tabulated independently and the max or min time that any one transaction spent from the start of queue to the end of service is not collected.

Statistics may be viewed on the screen during the running of the simulation and/or written to a file. The various statistics with definitions, and collection and display methods, are described in this Chapter.

6.1. ON SCREEN STATISTICS

Pedestal™ displays statistics for software and hardware components during the simulation by placing textual annotation beneath the icons on the diagram windows. As the values change, the annotation is updated on the diagram windows. Figure 6.1 shows an SCF window with statistics displayed while a simulation is running. Some statistics are available on almost every window in Pedestal™. The statistics under each icon refer to that icon and are calculated each time a transaction arrives at or departs from the icon. The statistics on connectors are user defined statistics that are calculated before a transaction crosses the connector. The display may be turned off from the Setup Menu; the "Show Runtime Statistics" menu item is a toggle that is initially checked. If this menu item is checked, statistics will be shown when the model while run; otherwise, the diagram statistics will be collected and calculated but not displayed. The decision whether to display statistics or not may be made during the simulation run by selecting the menu item.

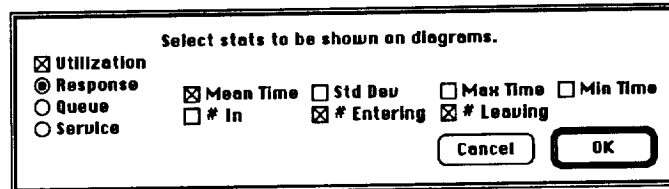


On Screen Statistics During Model Simulation
Figure 6.1

Due to the rapid rate at which the diagram statistics can change, there is a large overhead associated with displaying these statistics. The simulation will proceed considerably faster with the screen display off. The display may be used as a temporary diagnostic tool by viewing the statistics during a portion of the simulation to ascertain which hardware or software areas may be problematic, running the simulation to completion without the display, and then studying the problem areas from an output file. The simulation time is displayed in the upper right of the screen and can be used in conjunction with the display statistics to zero in on the simulation time at which a problem occurs. Queues can be seen building and response times seen degenerating. The detection of the where and when aspects of performance problems is greatly enhanced by the dynamic reporting of statistics in Pedestal™.

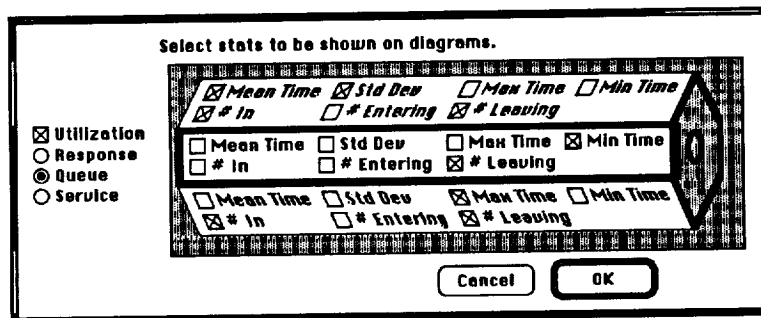
6.1.1. ICON STATISTICS

The "Pick Runtime Stats" item of the Setup Menu (Section 2.6) is used to specify the statistics, which if appropriate, will be displayed for the icons on the windows. Selecting this menu item will display the pick diagram statistics dialog shown in Figure 6.2.



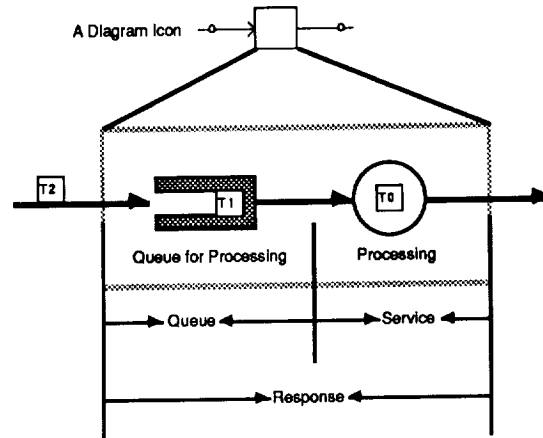
The Pick Diagram Statistics Dialog
Figure 6.2

Twenty two statistics can be specified from this dialog. The seven statistics shown in the middle of the dialog can be collected for response, queue, and service. The radio button currently selected determines which of the three groups of statistics is being set. You can think of the three groups of seven statistics check boxes as being on a polyhedron and the radio buttons determine which face of the polyhedron is visible in the dialog. This is demonstrated in Figure 6.3. The twenty second statistic is utilization.



Statistics "Polyhedron"
Figure 6.3

All icons process transactions and can be thought of as consisting of a queue of transactions awaiting processing within the icon, and the actual processing of the transactions. Statistics may be obtained concerning the queue and/or processing part of the icon. Statistics may also be reported from end to end, i.e. covering both the queue and processing. These three "processing regions" are called the queue, service, and response regions. The three radio buttons on the pick diagram statistics dialog allow the seven statistics to be specified for collection for each region. Figure 6.4 shows the decomposition of an icon and the processing regions.



Processing Zones
Figure 6.4

The statistics available on the dialog, their on screen abbreviation and their definitions are listed below. Note that all statistics are not available for all icons. For those statistics not implemented, <ni> will appear in the annotation beneath the icon.

Statistic	Abbrev.	Definition
Utilization	UTIL	Average Utilization of the icon = $\text{Sum}(\text{busy time}) / (\text{elapsed time} * \text{number of servers})$. The busy time is the amount of time for which a transaction was within the service region of the icon. The number of servers is one for all icons except device pool icons (Section 4.5.2). For these icons, the number of servers is the number of servers in the device pool.

Response time is the time a transaction spends in the response region of an icon.

Statistic	Abbrev.	Definition
Mean Response Time	mRT	Average of the response times.
SD Response Time	sRT	Standard deviation of the response times.
Maximum Response Time	xRT	Maximum response time recorded.
Minimum Response Time	nRT	Minimum response time recorded.

#In Response	#IN	Number of transactions currently in the response region of the icon.
#Entering Response	#AR	Total number of transactions which have arrived at the icon.
#Leaving Response	#DP	Total number of transactions which have left the icon.

Queue time is the time a transaction spends in the queue region of an icon.

Statistic	Abbrev.	Definition
Mean Queue Time	mQT	Average of the queue times.
SD Queue Time	sQT	Standard deviation of the queue times.
Maximum Queue Time	xQT	Maximum queue time recorded.
Minimum Queue Time	nQT	Minimum queue time recorded.
#In Queue	#iQT	Number of transactions currently in the queue of the icon.
#Entering Queue	#aQT	Total number of transactions which have arrived at the icon.
#Leaving Queue	#dQT	Total number of transactions which have left the icon.

Service time is the time a transaction spends in the service region of an icon.

Statistic	Abbrev.	Definition
Mean Service Time	mQT	Average of the service times.
SD Service Time	sQT	Standard deviation of the service times.
Maximum Service Time	xQT	Maximum service time recorded.
Minimum Service Time	nQT	Minimum service time recorded.
#In Service	#iQT	Number of transactions currently in the service region of the icon.
#Entering Service	#aQT	Total number of transactions which have arrived at the icon.
#Leaving Service	#dQT	Total number of transactions which have left the icon.

The SCF and RDM windows only display the response time statistics since none of the icons in these windows have an explicit queue component. The lock window displays all statistics except utilization, maximum response time and minimum response time which are not currently defined for locks. Statistics on the task window have two values separated by a slash ("/"). The value given before the slash corresponds to the task activations, the number after the slash to the queueing (buffer) for the task.

On hardware windows, the processor and memory icons have two aspects. The first aspect is the respective device, memory or processor. The

second aspect is the part of the device which performs transmissions. Thus, the icon statistics for these two icons have two numbers separated by a slash. The first number corresponds to the processor or memory, the second number corresponds to the associated transmission device.

6.1.2. CONNECTOR STATISTICS

User defined statistics may be displayed on screen for connectors on SCF windows. The connector statistics collection dialog, Figure 6.5 can be accessed by double clicking on a connector in an SCF window (Section 4.1.1.6). This dialog allows the user to name a statistic and define an expression which calculates that statistic. Additionally, if the show checkbox is checked for a defined statistic, that statistic will be displayed on screen during simulation run.

Label	Expression	Show	Log
Stat1	Dev[UTIL](*HW*,*cpu 1)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	0	<input type="checkbox"/>	<input type="checkbox"/>
	0	<input type="checkbox"/>	<input type="checkbox"/>
	0	<input type="checkbox"/>	<input type="checkbox"/>

Help Cancel OK

The Connector Statistics Dialog
Figure 6.5

These statistics will be calculated by evaluating the related expressions when a transaction crosses the associated connector. The values will be updated on the screen each time they are calculated. Note that use of the GetTParm and SetTParm functions in these expressions allows calculation of response times.

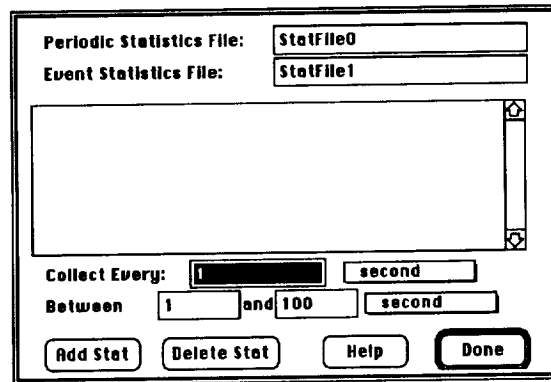
6.2. FILE STATISTICS

To make use of the high interoperability of the programs on the Macintosh™, Pedestal™ can write all statistics it collects to files in a format which many applications can read. These statistics can then be read by database, spreadsheet, and/or statistical analysis packages for analysis and graphing.

Two statistics files are generated. One contains all of the statistics calculated on connectors (Section 4.1.1.6) which are marked as "log" statistics. Since these statistics are generated in an undefined order and on unknown intervals, they are called event statistics. The other file contains periodic statistics. The periodic statistics are a set of statistics which are evaluated on a fixed time interval.

6.2.1. PERIODIC STATISTICS

Most often, statistics gathered on fixed time intervals throughout a simulation run are wanted. These statistics can be compared more readily since they are all defined for the same set of time values. The pick file stats dialog, shown in Figure 6.6, is used to define the set of periodic statistics to be gathered. This dialog is displayed by choosing the "Pick File Stats" command from the Setup Menu (Section 2.6).

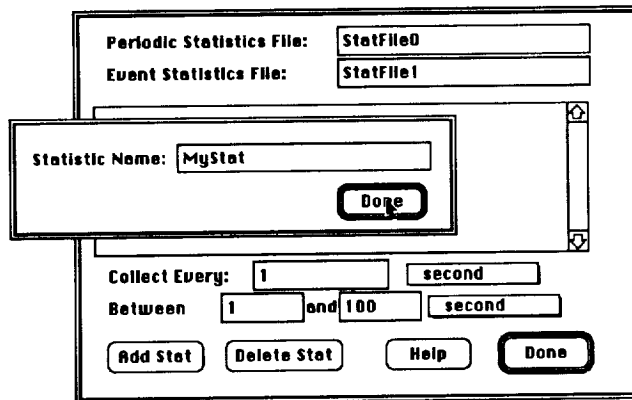


The Pick File Stats Dialog
Figure 6.6

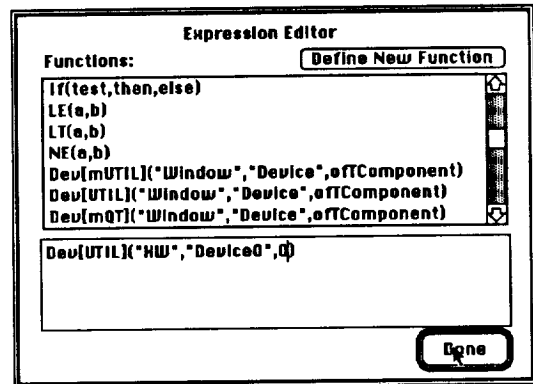
The "Periodic Statistics File" and "Event Statistics File" fields are used to specify the names of the files to be used to record the periodic and event statistics. These names must be distinct. Note that preexisting files with these names will be destroyed.

The "Collect Every: <number> <timeunit>" fields are used to set the collection interval for all periodic statistics. The popup menu is used for scaling. Likewise, the "Between <number> and <number> <timeunit>" fields are used to specify the part of the simulation during which periodic statistics are to be collected.

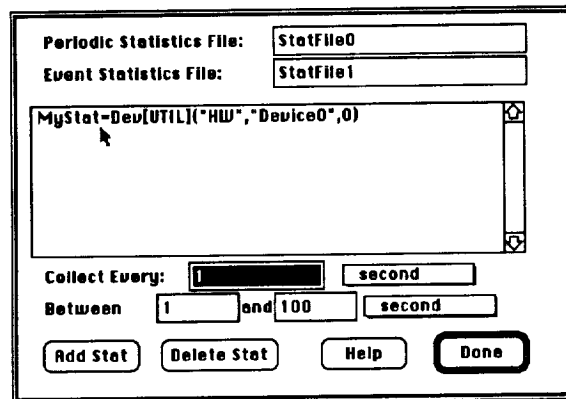
The list in the middle of the dialog is used to display all currently defined periodic statistics. To add a statistic to the list, click on the "Add Stat" button. A dialog will then ask for the name of the statistic as shown in Figure 6.7. This name is used as the column name in the periodic statistics file. After entering a name and clicking the "Done" button, the expression editor dialog is shown, as in Figure 6.8. For a discussion of the expression editor dialog see Section 3.1. Enter an expression which defines how the statistic is to be calculated and click on the "OK" button of the expression editor dialog. Now, as shown in Figure 6.9, the new statistic will appear in the statistic list of the pick file stats dialog.



The Statistic Name Dialog
Figure 6.7



The Expression Editor Dialog
Figure 6.8



New Statistic in the Statistic List
Figure 6.9

To edit an existing statistic, double click on its definition in the list. You will be taken through the same steps as when you defined it. First you will be allowed to change the name, then the expression, then the edited entry will be placed in the statistics list. To delete an existing statistic, click on the definition once to select it and click on the "Delete Stat" button.

6.2.2. EVENT STATISTICS

User defined statistics may be calculated and written to file whenever a transaction crosses a connector on an SCF window. The connector statistics collection dialog, Figure 6.10, can be accessed by double clicking on a connector in an SCF window (Section 4.1.1.6). This dialog allows the user to name a statistic and define an expression which calculates that statistic. Additionally, if the log checkbox is checked for a defined statistic, that statistic will be logged to a file each time it is calculated during a simulation run.

Label	Expression	Show	Log
Stat1	Dev[UTIL](*HW*, "cpu 1)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	0	<input type="checkbox"/>	<input type="checkbox"/>
	0	<input type="checkbox"/>	<input type="checkbox"/>
	0	<input type="checkbox"/>	<input type="checkbox"/>

Help Cancel OK

The Connector Statistics Dialog
Figure 6.10

These statistics will be calculated by evaluating the related expressions when a transaction crosses the associated connector. The values will be logged to the file each time they are calculated. Note that use of the GetTParm and SetTParm functions in these expressions allows evaluation of response times.

All logged connector statistics will be written to one file. The name of the file is determined by the "Event Statistics File" field of the pick file statistics dialog shown in Figure 6.11. This dialog is reached by selecting the "Pick File Stats" item of the report menu. A full discussion of this dialog appears in Section 6.2.1.

Periodic Statistics File: StatFile0

Event Statistics File: StatFile1

Collect Every: 1 second

Between 1 and 100 second

Add Stat Delete Stat Help Done

The Pick File Statistics Dialog
Figure 6.11

The format of this log file is:

Time	WindowName	FromIconName	ToIconName	Label	Value
------	------------	--------------	------------	-------	-------

The time field contains the simulation time at which the statistic was calculated. The window name, from icon name and to icon name are provided so that calculations of a statistic with the same name on different connectors can be differentiated if desired. The label gives the users name of the statistic from the connector statistics dialog. The value is the calculated value of the statistic. A tab separates each field since this is a form which most packages can import.

Since event statistics are calculated in an unknown order and on an unknown and irregular time interval, the file will require sorting to get all the values related to a specific label together. Also, the FromIconName and ToIconName are supplied since Label names are not necessarily unique. This information, while cumbersome, allows exact identification of the statistic which was calculated.

6.2.3. IMPORTING FILE STATISTICS INTO OTHER APPLICATIONS

Any data manipulation package which can import tab delimited text files can import the statistics files generated by Pedestal™. The periodic statistics defined in the pick file statistics dialog (Section 6.2.1) will be written to a file. The values will be in columns, and the first row will contain the statistics names. The first column will contain the simulation time at which each row was calculated.

The connector statistics which are logged will appear in one file. Because these statistics are not calculated at fixed intervals and not necessarily all at the same time, the file may require some sorting once imported into a package to get the data in the proper form for your analysis. This file will contain all information necessary to sort the information in any manner you wish.

Refer to the manual for the package(s) you wish to use to determine the method of importing data files. The following Sections detail the steps required to import Pedestal™ statistics files into Excell™ and Exstatics™.

6.2.3.1. EXCELL™

Microsoft Excell™ is a popular spreadsheet package on the Macintosh™. The following is an example of how to import a Pedestal™ statistics file into Excell™ and make a chart. Note that for charting of "event" statistics (from statistics defined on connectors) Excell™ is not recommended since it does not support uneven intervals along the X (category) axis. Excell™ does provide a powerful and simple sorting mechanism which is well suited to reordering the data in the event statistics file.

The numbers in this example were generated by a short simulation of a simple model which had two processors. Figure 6.12 shows the pick file statistics dialog with the definition of the statistics which were used.

Periodic Statistics File: StatFile0
 Event Statistics File: StatFile1

Util(cpu0)=Dev[UTIL](*HW*, "Device0")
 Util(cpu1)=Dev[UTIL](*HW*, "Device1")

Collect Every: 0.1 second
 Between 1 and 100 second

Add Stat Delete Stat Help Done

Figure 6.12.

Once you have run the simulation and produced your statistics file, exit Pedestal™ and launch Excell™. Choose the "Open" command from the Excell™ file menu. This will display the standard Macintosh™ open file dialog; use this dialog to get to the folder which contains the statistics file and open it. A new worksheet will be created with the same name as the statistics file. Figure 6.13 shows such a worksheet.

	A	B	C	D	E	F
1	Time	Util(cpu0)	Util(cpu1)			
2	1	0.700023	0			
3	1.5	0.800015	0.133349			
4	2	0.850011	0.350011			
5	2.5	0.866179	0.480009			
6	3	0.888482	0.566674			
7	3.5	0.904414	0.618699			
8	4	0.916362	0.666362			
9	4.5	0.925655	0.703433			
10	5	0.933089	0.733089			
11	5.5	0.909091	0.757354			
12	6	0.833333	0.777575			
13	6.5	0.769231	0.769231			
14	7	0.754009	0.714286			
15	7.5	0.770408	0.666667			
16	8	0.75	0.659758			
17	8.5	0.705882	0.679772			
18	9	0.666667	0.666667			
19	9.5	0.631579	0.631579			
20	10	0.6	0.6			

Figure 6.13.

Note that the data series are in columns with titles in the first row. Also, the first column contains the X axis values (categories). This worksheet may be manipulated like any other in Excell™. To chart the data, select the data and choose "Copy" from the edit menu. Next, create a new chart. Now, select "Paste Special" from the edit menu. This will display the paste special dialog; edit this dialog such that its contents match that of the dialog in Figure 6.14. Click OK.

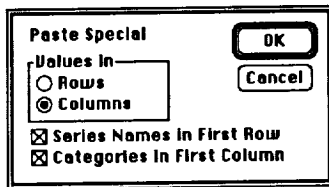


Figure 6.14.

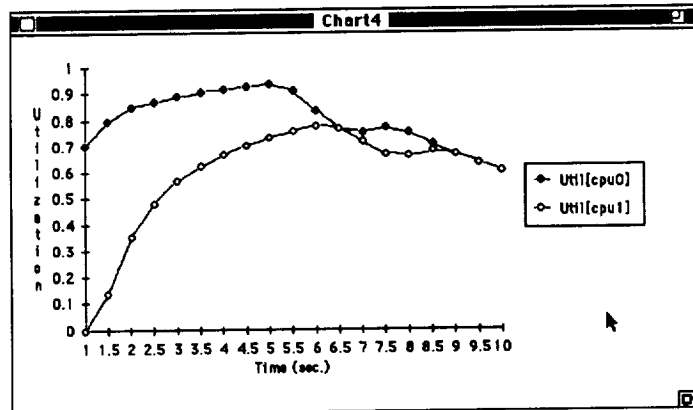


Figure 6.15.

The data will now be charted. You can use the Excell™ commands to change the graph type, add legends, etc.. Figure 6.15 shows a sample chart produced by Excell™.

6.2.3.2. EXSTATISTICS™

Exstatistics™ is a statistical analysis package for the Macintosh™. The following is an example of how to import a Pedestal™ statistics file into Exstatistics™. Unlike Excell™, Exstatistics™ does support uneven intervals along the X axis. However, sorting is limited.

The numbers in this example were generated by a short simulation of a simple model which had two processors. Figure 6.16 shows the pick file statistics dialog with the definition of the statistics which were used.

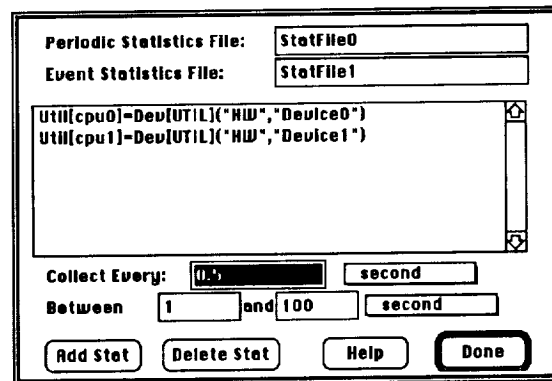


Figure 6.16.

Once you have run the simulation and produced your statistics file, exit Pedestal™ and launch Exstatistics™. Chose the "Open" command from Exstatistics™s file menu. This will display the standard Macintosh™ open file dialog, use this dialog to get to the folder which contains the statistics file and open it. A dialog will be displayed to allow you to tell Exstatistics™ how to read the file. For periodic file statistics set the dialog buttons as shown in Figure 6.17.

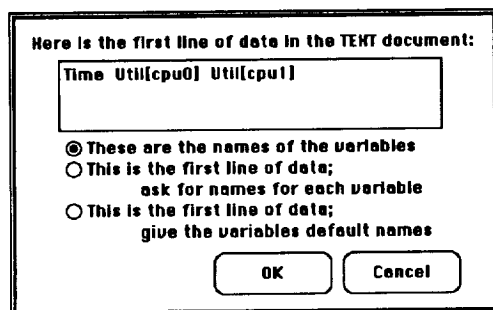


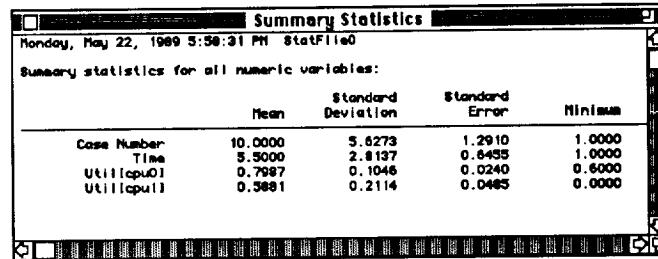
Figure 6.17

	Time	Util(cpu0)	Util(cpu1)	New Variable
1	1	0.700023	0	
2	1.5	0.800015	0.133349	
3	2	0.850011	0.350011	
4	2.5	0.866179	0.480009	
5	3	0.888482	0.566674	
6	3.5	0.904414	0.618699	
7	4	0.916362	0.666362	
8	4.5	0.925655	0.703433	
9	5	0.933089	0.733089	
10	5.5	0.909091	0.757354	
11	6	0.833333	0.777575	
12	6.5	0.769231	0.769231	
13	7	0.754009	0.714286	
14	7.5	0.770408	0.666667	
15	8	0.75	0.659758	
16	8.5	0.795882	0.679772	

Figure 6.18

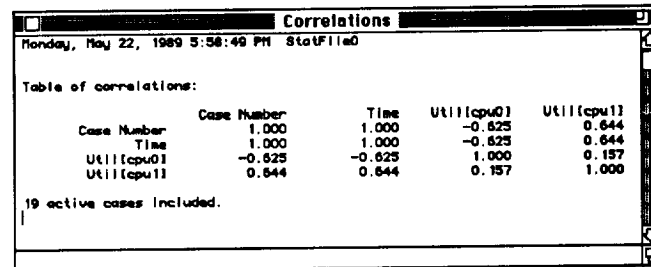
For event statistics, click either the second or third radio button. Exstatics™ will read the file and create a new data set window. Figure 6.18 is an example of this window.

The data may now be analyzed and/or charted. Figures 6.19 and 6.20 show a few simple reports generated by Exstatics™ relating to the example data set.



	Mean	Standard Deviation	Standard Error	Minimum
Case Number	10.0000	5.6273	1.2910	1.0000
Time	5.5000	2.8137	0.6435	1.0000
Util(cpu0)	0.7987	0.1046	0.0240	0.6000
Util(cpu1)	0.5681	0.2114	0.0485	0.0000

Figure 6.19



	Case Number	Time	Util(cpu0)	Util(cpu1)
Case Number	1.000	1.000	-0.625	0.644
Time	1.000	1.000	-0.625	0.644
Util(cpu0)	-0.625	-0.625	1.000	0.157
Util(cpu1)	0.644	0.644	0.157	1.000

19 active cases included.

Figure 6.20

7. RUNNING A MODEL

Several requirements must be satisfied before simulation of a Pedestal™ may begin. Most of these requirements involve the specification of software grouping and assignment to hardware. The grouping may be done by specifying tasks on the module definitions or by selection the Module->Task item from the Map Menu. Since the hardware and software designs are independent, the mappings which relate the software and hardware must be done by menu selection. Other requirements pertain to how the diagrams which describe the software and hardware are drawn. These requirements ensure that the diagrams are syntactically correct.

7.1. MAPPING REQUIREMENTS

The mechanics of mapping are explained in Chapter 5 and it is important to understand that all modules must be assigned to task, all tasks must be assigned to processors, and all datastores must be mapped to memory devices. Upon selection from the menu of a particular mapping, the association list (Chapter 5) specific to that mapping appears. Pedestal™ aids the user by displaying only those items appropriate to that type of mapping.

In the event that the mappings are incomplete at the time the run command is issued, the dialog shown in Figure 7.1 appears. Upon dismissal of the dialog, the deficient mapping dialog, shown in Figure 7.2, comes up automatically for completion. Selection of the Cancel button will retract the command to run. This would be necessary in cases where there were elements lacking that are necessary for completion of mapping; e.g., if module to task mapping were incomplete and there were no tasks currently defined, the user would not be able to rectify the error from the association list. Cancellation of the run command puts the user back into the model definition mode where tasks may be defined from either module icons or from the task window.

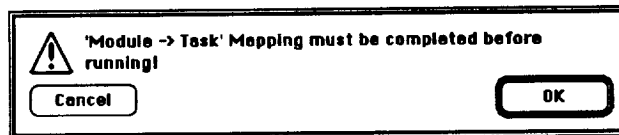


Figure 7.1
Incomplete Mapping Dialog

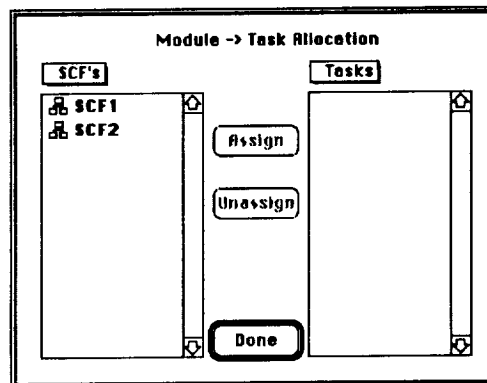


Figure 7.2
Attempt to Map with No Classes

The example cited above is for module->task mapping. Parallel situations may arise for the other mappings. New devices, processors or memories, are created in the HW or Control windows; managers are not part of the mandatory mapping but they may be created in the manager window, accessible from the Desktop window.

7.2. SOFTWARE WINDOW REQUIREMENTS

In order to achieve a successful simulation there must be complete flows in the SCF, Macro, and RDM windows. Each window must have one and only one each stimulus and terminator. All flows originate in the stimulus and terminate in the terminator. The branching creates parallel or exclusive alternate flows but all icons with the exception of the stimulus and terminator must have an incoming arrow and an outgoing arrow in order for the model to be complete.

7.3. HARDWARE WINDOW REQUIREMENTS

The hardware and node window requirements differ from the software windows in that connections are not required. It is possible to have a valid Pedestal™ model in which all hardware devices are free standing. Routing accesses must be explicitly expressed by connecting components. For example a processor that is in close proximity to a bus will not use that bus unless it is connected to it in the diagram.

GLOSSARY

accessory: See **desk accessory**, **peripheral device**.

active window: The frontmost **window** on the **desktop**; the window where the next action will take place. An active window's **title bar** is highlighted. (M)

alert: A warning or report of an error in the form of an alert box, a sound from the computer's speaker, or both. (M)

alert box: A box that appears on the screen to give a warning or to report an error message during use of an application. (M)

algorithm: A step-by-step procedure for solving a problem or accomplishing a task.

American Standard Code for Information Interchange: See **ASCII**.

ANSI: Acronym for *American National Standards Institute*, which sets standards for many technical fields and is the most common standard for computer terminals.

Apple key: A key marked with an outlined Apple symbol; on older Apple II machines, it's called *Open Apple*. On some keyboards, the Apple key also acts as the Command key and carries both the Apple symbol and the propeller-shaped Command symbol.

Apple menu: The menu farthest to the left in the **menu bar**, indicated by an Apple symbol, from which you choose **desk accessories**. (M)

application program: A program written for some specific purpose, such as word processing, data base management, graphics, or telecommunication. Compare **system program**.

application software: A collective term for **application programs**.

arithmetic expression: A combination of numbers and arithmetic operators (such as $3 + 5$) that indicates some operation to be carried out.

arithmetic operation: One of the five actions computers can perform with numbers: addition, subtraction, multiplication, division, and exponentiation.

arithmetic operator: An operator, such as $+$, that combines numeric values to produce a numeric result. Compare **logical operator**, **relational operator**.

array: An ordered collection of information of a given, defined type. Each element of the array can be referred to by a numerical subscript.

ASCII: Acronym for *American Standard Code for Information Interchange*, pronounced "ASK-ee." A code in which the numbers from 0 to 127 stand for text characters. ASCII code is used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device. Compare **EBCDIC**.

Backspace key: A key that backspaces over and erases the previously typed character or the current selection.

back up: (v) To make a spare copy of a disk or of a file on a disk. Backing up your files and disks ensures that you won't lose information if the original is lost or damaged.

backup: (n) A copy of a disk or of a file on a disk. It's a good idea to make backups of all your important disks and to use the copies for everyday work, keeping the originals in a safe place. (Some program or startup disks cannot be copied.)

bit: A contraction of *binary digit*. The smallest unit of information that a computer can hold. The value of a bit (1 or 0) represents a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing. See also **binary system**.

bit rate: The speed at which bits are transmitted, usually expressed as *bits per second*, or *bps*. Compare **baud**.

bits per second: See **bit rate**.

bps (bits per second): See **bit rate**.

buffer: A “holding area” of the computer’s memory where information can be stored by one program or device and then read at a different rate by another; for example, a print buffer. In editing functions, an area in memory where deleted (cut) or copied data is held. In some applications, this area is called the *Clipboard*. See also **type-ahead buffer**.

button: A pushbutton-like image in dialog boxes where you click to designate, confirm, or cancel an action. See also **mouse button**.

byte: A unit of information consisting of a fixed number of **bits**. On Apple II systems, one byte consists of a series of eight bits, and a byte can represent any value between 0 and 255. The sequence represents an instruction, letter, number, punctuation mark, or other character. See also **kilobyte**, **megabyte**.

Cancel button: A button that appears in a dialog box. Clicking it cancels the command.

Caps Lock key: A key that, when engaged, causes subsequently typed letters to appear in uppercase; its effect is like that of the Shift key except that it doesn’t affect numbers and other non-letter symbols.

carriage return: An ASCII character (decimal 13) that ordinarily causes a printer or display device to place the next character on the left margin.

case sensitive: (adj) Able to distinguish between uppercase characters and lowercase characters. Programming languages are *case sensitive* if they require all-uppercase letters, all-lowercase letters, or proper use of uppercase and lowercase. For example, Applesoft BASIC recognizes only uppercase. Instant Pascal, on the other hand, is not case sensitive; you can use any combination of uppercase and lowercase letters you like.

character keys: Keys on a computer keyboard—such as letters, numbers, symbols, punctuation marks—used to generate text or to format text; any key except Shift, Caps Lock, Command, Option, Open Apple, Solid Apple, Control, and Escape. Character keys repeat when you press and hold them down.

check box: A small box or circle associated with an option in a dialog box. When you click the check box, you may change the option or affect related options.

choose: To pick a command by dragging through a menu. You often choose a command after you’ve selected something for the program to act on. (M)

click: To position the pointer on something, and then to press and quickly release the mouse button. (M)

Clipboard: The holding place for what you last cut or copied; a buffer area in memory. Information on the Clipboard can be inserted (pasted) into documents.

close: To turn a window back into the icon that represents it. (M)

close box: The small white box on the left side of the **title bar** of an active window. Clicking it closes the window.

command: An instruction that causes the computer to perform some action. A command can be typed from a keyboard, selected from a menu with a hand-held device (such as a mouse), or embedded in a program.

command code: One or more characters whose function is to change the way a program or device acts (as opposed to text, which is simply printed).

Command key: A key that, when held down while another key is pressed, causes a command to take effect. When held down in combination with dragging the mouse, the Command key lets you drag a window to a new location without activating it. The Command key is marked with a propeller-shaped symbol. On some machines, the Command key has both the propeller symbol and the Apple symbol on it. (M)

conditional branch: A branch whose execution depends on the truth of a condition or the value of an expression. Compare **unconditional branch**.

context sensitive: (adj) Able to perceive the situation in which an event occurs. For example, an application program might present help information specific to the particular task you're performing, rather than a general list of commands; such help would be context sensitive.

control key: A general term for a key that controls the operation of other keys; for example, Caps Lock, Command, Control, Open Apple, Option, and Shift. When you hold down or engage a control key while pressing another key, the combination makes that other key behave differently. Also called a *modifier key*.

cursor: A symbol displayed on the screen marking where the user's next action will take effect or where the next character typed from the keyboard will appear. (H)

cut: To remove something by selecting it and choosing Cut from a menu. What you cut is placed on the Clipboard. (M) In other editing applications, delete serves the same function. See also **buffer**.

data: Information, especially information used or operated on by a program. The smallest unit of information a computer can understand is a **bit**.

data bits: The bits in a communication transfer that contain information. Compare **start bit**, **stop bit**.

default: A preset response to a question or prompt. The default is automatically used by the computer if you don't supply a different response. Default values prevent a program from stalling or crashing if no value is supplied by the user.

delete: To remove something, such as a character or word from a file, or a file from a disk. Keys such as the Backspace key and the Delete key can remove one character at a time by moving to the left. In the Macintosh family, the Cut command removes selected text and places it on the Clipboard; the Clear command and the Backspace key remove selected text without placing it on the Clipboard.

Delete key: A key on the upper-right corner of the Apple IIe and IIc keyboards that erases the character immediately preceding (to the left of) the cursor. Similar to the Macintosh Backspace key.

delimiter: A character that is used for punctuation to mark the beginning or end of a sequence of characters, and which therefore is not considered part of the sequence itself. For example, Applesoft BASIC uses the double quotation mark (") as a delimiter for string constants: the string "DOG" consists of the three characters D, O, and G, and does not include the quotation marks.

desktop: In Macintosh applications, the computer's working environment—the menu bar and the gray area on the screen. You can have a number of documents on the desktop at the same time. In AppleWorks, the Desktop is an area of memory where you can keep several files at a time. Once files are on the Desktop, you can switch back and forth between files without having to get files from the data disk.

dialog box: A box that contains a message requesting more information from you. Sometimes the message warns you that you're asking your computer to do something it can't do or that you're about to destroy some of your information. In these cases the message is often accompanied by a beep. (M)

double-click: To position the pointer where you want an action to take place, and then press and release the mouse button twice in quick succession without moving the mouse. (M)

drag: To position the pointer on something, press and hold the mouse button, move the mouse, and release the mouse button. When you release the mouse button, you either confirm a selection or move an object to a new location. (M)

edit: To change or modify. For example, to insert, remove, replace, or move text in a document.

Enter key: A key that confirms or terminates an entry or sometimes a command.

error condition: The state of the hardware or program after it has detected a fault in one or more commands sent to it.

error message: A message displayed or printed to tell you of an error or problem in the execution of a program or in your communication with the system; an error message is often accompanied by a beep.

expression: A formula in a program that defines a calculation to be performed.

file: Any named, ordered collection of information stored on a disk. Application programs and operating systems on disks are examples of files. You make a file when you create text or graphics, give the material a name, and save it to disk; in this sense, synonymous with **document**.

file management: A general term for copying files, deleting files, and other chores involving the contents of disks.

filename: The name that identifies a file. The maximum character length of a filename and the rules for naming a file vary under different operating systems. Compare **pathname**. (II)

Finder: An application that's always available on the **desktop**. You use it to manage documents and applications, and to get information to and from disks. (M)

fixed-point notation: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the number. Typically, the point is considered to lie at the right end of the number so that the number is interpreted as an integer. Compare **floating-point notation**.

flag: A variable whose value (usually 1 or 0, standing for *true* or *false*) indicates whether some condition holds or whether some event has occurred. A flag is used to control the program's actions at some later time.

floating-point notation: A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to "float" to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point notation**.

folder: A holder of documents and applications on the desktop. Folders, like subdirectories, allow you to organize information in any way you want. (M)

function: A preprogrammed calculation that can be carried out on request from any point in a program. Because a function takes in one or more arguments and returns a single value, it can be embedded in an expression.

graph: A pictorial representation of data.

graphics: (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

hard copy: Information printed on paper, as opposed to being stored on disk.

icon: An image that graphically represents an object, a concept, or a message. For example, an unopened MacWrite document looks like a sheet of paper with lines like writing on it; an unopened MacPaint document looks like a sheet of paper with a paint brush painting a line.

input: (n) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. Compare **output**.

input/output (I/O): The process by which information is transferred between the computer's memory and its keyboard or peripheral devices.

insertion point: The place in a document where something will be added; it is selected by clicking and is represented by a blinking vertical bar.

installation: The process of adding information to the System file of a disk. For example, the Printer Installer on the *LaserWriter Installation Disk* installs the LaserWriter and new ImageWriter software.

interactive: (adj) Operating by means of a dialog between the computer system and a human user.

interface: (1) The point at which independent systems or diverse groups interact. The devices, rules, or conventions by which one component of a system communicates with another. Also, the point of communication between a person and a computer. (2) The part of a program that defines constants, variables, and data structures, rather than procedures.

interrupt: A temporary suspension in the execution of a program that allows the computer to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

I/O device: Input/output device. A device that transfers information into or out of a computer. See **input**, **output**, **peripheral device**.

K: See **kilobyte**.

kilobyte (K): A unit of measurement consisting of 1024 (2^{10}) **bytes**. In this usage, *kilo* (from the Greek, meaning a thousand) stands for 1024. Thus, 64K memory equals 65,536 bytes. See also **megabyte**.

leading zero: A zero occurring at the beginning of a decimal number; deleted by most computing programs.

Macintosh: A family of Apple computers; for example, the Macintosh 512K and the Macintosh Plus. Macintosh computers have high-resolution screens and use mouse devices for choosing commands and for drawing pictures.

megabyte (Mb): A unit of measurement equal to 1024 kilobytes, or 1,048,576 bytes. See **kilobyte**.

menu: A list of choices presented by a program, from which you can select an action. In Macintosh, menus appear when you point to and press menu titles in the **menu bar**. Dragging through the menu and releasing the mouse button while a command is highlighted chooses that command.

menu bar: The horizontal strip at the top of the screen that contains menu titles. (M)

menu title: A word, phrase, or icon in the menu bar that designates one menu. Pressing on the menu title causes the title to be highlighted and its menu to appear below it. (M)

microsecond (ms): One millionth of a second.

millisecond (ms): One thousandth of a second.

mouse: A small device you move around on a flat surface next to your computer. The mouse controls a pointer on the screen whose movements correspond to those of the mouse. You use the pointer to select operations, to move data, and to draw with in graphics programs.

mouse button: The button on the top of the mouse. In general, pressing the mouse button initiates some action on whatever is under the pointer, and releasing the button confirms the action.

multitasking: A process that allows a computer to perform two or more tasks during a given period of time; it is accomplished by alternating the actions of the computer between tasks.

nanosecond (ns): One billionth of a second.

network: A collection of interconnected, individually controlled computers, together with the hardware and software used to connect them. A network allows users to share data and peripheral devices such as printers and storage media, to exchange electronic mail, and so on.

open: To make available. You open files or documents in order to work with them. In Macintosh, when you double-click an icon or select it and choose the Open command, you cause a window with the contents of that icon to come into view. You may then perform further actions in the window, if it's an active window.

Option key: A modifier key that gives a different meaning or action to another key you type or mouse actions you perform. You use it to type foreign characters or special symbols contained in the optional character set. (M)

output: (n) Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem. Compare **input**.

paste: To place the contents of the Clipboard—whatever was last cut or copied—at the insertion point.

peripheral: (adj) At or outside the boundaries of the computer itself, either physically (as a peripheral device) or logically (as a peripheral card). (n) Short for *peripheral device*.

pipelining: A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All else being equal, a processor with this feature runs faster than one without it.

precedence: The order in which operators are applied in evaluating an expression. Precedence varies from language to language, but usually resembles the precedence rules of algebra.

press: (1) To position the pointer on something and then hold down the mouse button without moving the mouse. (2) To strike a key and then release it; you hold a key down only if you want to repeat a character (or you are using a control key with another key).

prompt: A message on the screen that tells you of some need for response or action. A prompt usually takes the form of a symbol, a message, a dialog box, or a menu of choices. (II)

prompt character: A text character displayed on the screen, usually just to the left of a **cursor**, where your next action is expected. The prompt character often identifies the program or component of the system that's prompting you. For example, Applesoft BASIC uses a square bracket prompt character (**[**); Integer BASIC, an angle bracket (**>**); and the system Monitor program, an asterisk (*****). (II)

protocol: A formal set of rules for sending and receiving data on a communication line. For example, binary synchronous communications (BSC) is a protocol.

pseudo-random numbers: A sequence of numbers, determined by some defined arithmetic process, that is satisfactorily close to a true random sequence for a given purpose. Microcomputers can generate pseudo-random numbers, and thus can simulate games of chance, such as dice-based games and card games.

queue: A list in which entries are added at one end and removed at the other, causing entries to be removed in first-in, first-out (FIFO) order. Compare **stack**.

read: To transfer information into the computer's memory from a source outside the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

real number: In computer usage, a number that may include a fractional part; represented inside the computer in floating point notation. Because a real number is of infinite precision, this representation is usually approximate. Compare **integer**.

Return key: A key that causes the cursor or insertion point to move to the beginning of the next line. It's also used in some cases to confirm a command.

run: (1) To execute a program. When a program runs, the computer performs the instructions. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

save: To store information by transferring the information from main memory to a disk. Work not saved disappears when you turn off the computer or when the power is interrupted.

scientific notation: A method of expressing numbers in terms of powers of ten, useful for expressing very small or very large numbers.

For example, 6.02E23, means 6.02 times ten

to the 23rd power. (The letter *E* stands for exponent.) The number is easier to understand in this form than in the form 60200000000000000000000. Applesoft BASIC

uses this method to display real (floating-point) numbers with more than nine digits.

scroll arrow: An arrow at either end of a scroll bar. Clicking a scroll arrow moves a document or directory one line. Pressing a scroll arrow moves a document continuously. (M)

scroll bar: A rectangular bar that may be along the right or bottom of a window. Clicking or dragging in the scroll bar causes your view of the document to change. (M)

scroll box: The white box in a scroll bar. The position of the scroll box in the scroll bar indicates the position of what's in the window relative to the entire document. (M)

seed: A value used to begin a repeatable sequence of **pseudo-random numbers**.

select: To designate where the next action will take place. To select using a mouse, you click an icon or drag across information. You can also select menu items by typing a letter or number at a prompt, by using a combination keypress, or by using arrow keys.

selection: The information or items that will be affected by the next command. The selection is usually highlighted.

Shift-click: A technique that allows you to extend or shorten a selection by positioning the pointer at the end of what you want to select and holding down the Shift key while clicking the mouse button.

Shift-drag: A technique that allows you to select multiple objects by holding down the Shift key while you drag diagonally to enclose the objects in a rectangle.

Shift key A key that, when pressed, causes the subsequent letter you type to appear in uppercase, or the top symbol on a two-character key to be produced.

simulation: A computerized representation of some process in action—for example, a flight simulation.

size box: A box on the bottom-right corner of some active windows. Dragging the size box resizes the window.

stack: A list in which entries are added (pushed) or removed (popped) at one end only (the top of the stack), causing them to be removed in last-in, first-out (LIFO) order. Compare **queue**.

Standard Apple Numeric Environment (SANE): The set of methods that provides the basis for floating-point calculations in Apple computers. SANE meets all requirements for extended-precision, floating-point arithmetic as prescribed by IEEE Standard 754 and ensures that all floating-point operations are performed consistently and return the most accurate results possible.

start up: To get the system running. Starting up is the process of first reading an operating system program from the disk, and then running an application program.

subdirectory: A directory within a directory; a file containing the names and locations of other files.

syntax: (1) The rules governing the structure of statements or instructions in a programming language. (2) A representation of a command that specifies all the possible forms the command can take.

system: A coordinated collection of interrelated and interacting parts organized to perform some function or achieve some purpose—for example, a computer system comprising a processor, keyboard, monitor, and disk drive.

system software: The component of a computer system that supports application programs by managing system resources such as memory and I/O devices.

time sharing: Sharing a computer between two or more users, usually through multitasking.

title bar: The horizontal bar at the top of a window that shows the name of the window's contents. You can move the window by dragging the title bar. (M)

token: An abbreviation of a string of characters. For example, Applesoft BASIC stores commands internally as single-character tokens.

unconditional branch: A branch that does not depend on the truth of any condition. Compare **conditional branch**.

user: A person operating or controlling a computer system.

user interface: The rules and conventions by which a computer system communicates with the person operating it.

value: An item of information that can be stored in a variable, such as a number or a string.

variable: (1) A location in the computer's memory where a value can be stored. (2) The symbol used in a program to represent such a location.

window: (1) The area that displays information on a desktop; you view a document through a window. You can open or close a window, move it around on the desktop, and sometimes change its size, scroll through it, and edit its contents. (M) (2) The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen. Compare **viewport**.

write: To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

zoom box: A small box with a smaller box enclosed in it found on the right side of the **title bar** of some windows. Clicking the zoom box expands the window to its maximum size; clicking it again returns the window to its original size.



Report Documentation Page

1. Report No.		2. Government Accession No.		3. Recipient's Catalog No.	
4. Title and Subtitle Final Report				5. Report Date 29 June 89	
				6. Performing Organization Code	
7. Author(s)				8. Performing Organization Report No. Deliverable 0002	
				10. Work Unit No.	
9. Performing Organization Name and Address Advanced System Technologies, Inc. 12200 E. Briarwood Ave. Suite 260 Englewood, CO 80112				11. Contract or Grant No. NAS7-995	
				13. Type of Report and Period Covered Final Report 3/87-6/29/89	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001 NASA Resident Office - Jet Propulsion Laboratory				14. Sponsoring Agency Code	
15. Supplementary Notes					
16. Abstract This report summarizes Advanced System Technologies' accomplishments on the Phase II SBIR contract NAS7-995. Project summary, objectives, work carried out, and results obtained are presented.					
17. Key Words (Suggested by Author(s)) SBIR, Status			18. Distribution Statement Unclassified - Unlimited		
19. Security Classif. (of this report) Unclassified		20. Security Classif. (of this page) Unclassified		21. No. of pages 232	
				22. Price	